U.S.N.A --- Trident Scholar project report; no. 368 (2008)

A Parallel Implementation of Three-Dimensional, Lagrangian Shallow Water Equations

by

Midshipman 1/C Daniel D. Hartig
United States Naval Academy
Annapolis, Maryland

_____
(signature)

Certification of Advisers Approval

Visiting Research Professor J. M. Greenberg

_____
(signature)

_____
(date)

Professor Reza Malek-Madani
Mathematics Department

_____
(signature)

_____
(date)

Associate Professor Chris Brown
Computer Science Department

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research and Scholarship

_____
(signature)

_____
(date)

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	7 May 2008	

4. TITLE AND SUBTITLE
A Parallel Implementation of Three-Dimensional Lagrangian Shallow Water Equations

5. FUNDING NUMBERS

6. AUTHOR(S)

Hartig, Daniel D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

US Naval Academy
Annapolis, MD 21402

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

Trident Scholar project report no. 368 (2008)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT (cont from p.1) The next stage of the project jumped forward to a serial three-dimensional implementation of the shallow water equations developed by Dr. Greenberg. This model is similar to the two-dimensional model but with the added complexity of two lateral degrees of freedom— both the x- and y-directions—while the vertical component is still treated the same as in the two-dimensional model. In order to split the two-dimensional domain grid into a set of smaller domains assigned to the various processors, the special geometry of that grid had to be taken into account. Once a scheme was in place to divide the grid while maintaining all of its special properties, the computation on each sub-domain was performed with the same program that operates on the entire domain. This allowed easy implementation of a parallel solution: each node ran a modified serial implementation on a subsection of the larger problem that had been carefully separated from the whole. A process was created to allow the nodes to communicate data from the edges of their sub-domains as they advanced forward through time. There are two deliverable products from this project. First, there is a serial two-dimensional model of fluid circulation that takes into account many different user-designated initial conditions and can be useful for determining how well the mathematics of this model can approximate physical phenomena. Secondly, this project produced a three-dimensional parallel model that serves as a proof of concept for future development of more advanced parallel models.

14. SUBJECT TERMS shallow water equations, free boundary, Lagrangian, s-Transformation, coastal ocean circulation model, MATLAB

15. NUMBER OF PAGES
84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Abstract

This project developed fluid circulation models for the two- and three-dimensional Lagrangian shallow water equations. There were two stages to this development: in the first, the two-dimensional shallow water equations were transformed from first principles of oceanography into a serial implementation in MATLAB. In the second part, a serial implementation of the three-dimensional shallow water equations, developed by Dr. James Greenberg, was modified to run in parallel on many nodes of a computing cluster.

The serial, one-dimensional model includes one lateral degree of freedom—the x-direction. The vertical or z-direction is modeled by layers; velocity in this direction was removed by a series of transformations to the governing equations. Development started with the conservation of mass and conservation of momentum equations. The traction terms in these equations were approximated using a method previously established by Mellor and Blumberg in their work on the Princeton Ocean Model (POM). These equations were scaled and then transformed to an s-coordinate system, again following the established method of POM. Once a set of partial differential equations were derived, these equations were discretized in space and time and solved in MATLAB. The implementation of the model in MATLAB allows the user a wide range of initial conditions for factors such as the bathymetry, initial area covered in fluid, magnitude of friction coefficients, and more. For a given set of initial conditions, the model steps forward in time by user-designated time steps solving for position, velocity, and depth of the fluid at each step and visually representing this information with appropriate graphs.

The next stage of the project jumped forward to a serial three-dimensional implementation of the shallow water equations developed by Dr. Greenberg. This model is similar to the two-dimensional model but with the added complexity of two lateral degrees of freedom—both the x- and y-directions—while the vertical component is still treated the same as in the two-dimensional model. In order to split the two-dimensional domain grid into a set of smaller domains assigned to the various processors, the special geometry of that grid had to be taken into account. Once a scheme was in place to divide the grid while maintaining all of its special properties, the computation on each sub-domain was performed with the same program that operates on the entire domain. This allowed easy implementation of a parallel solution: each node ran a modified serial implementation on a subsection of the larger problem that had been carefully separated from the whole. A process was created to allow the nodes to communicate data from the edges of their sub-domains as they advanced forward through time.

There are two deliverable products from this project. First, there is a serial two-dimensional model of fluid circulation that takes into account many different user-designated initial conditions and can be useful for determining how well the mathematics of this model can approximate physical phenomena. Secondly, this project produced a three-dimensional

parallel model that serves as a proof of concept for future development of more advanced parallel models.

Key Words: shallow water equations, free boundary, Lagrangian, s-Transformation, coastal ocean circulation model, MATLAB

Contents

List of Figures

List of Tables

List of Variables

$\hat{x}$, $\hat{y}$, $\hat{z}$: Directions in the Cartesian coordinate system.

$X(\hat{t})$, $Y(\hat{t})$, $Z(\hat{t})$: Cartesian components of particle location in Lagrangian representation.

s: Coordinate to which the Cartesian z is transformed. s is 0 on the bottom surface of a fluid and 1 on the upper surface.

$\hat{u}(\hat{x}, \hat{y}, \hat{z}, \hat{t})$: Physical velocity in the x direction in Eulerian representation.

$\hat{v}(\hat{x}, \hat{y}, \hat{z}, \hat{t})$: Physical velocity in the y direction in Eulerian representation.

$\hat{w}(\hat{x}, \hat{y}, \hat{z}, \hat{t})$: Physical velocity in the z direction in Eulerian representation.

ρ: Density of the fluid. Assumed to be a constant equal to 1.

$\hat{p}(\hat{x}, \hat{y}, \hat{z}, \hat{t})$: Pressure of the fluid.

f: Coriolis force.

g: Acceleration due to gravity.

$\hat{\sigma}_{ab}$: Traction on an elemental cube, the force is acting on face of the cube a in direction b. The numbers 1, 2, and 3, correspond to directions x, y, and z.

E: Eddy viscocity.

$\bar{u}$, $\bar{v}$, $\bar{w}$, $\bar{p}$: Non-dimensionalized forms of $\hat{u}$, $\hat{v}$, $\hat{w}$, and $\hat{p}$; each is a function of $\bar{x}$, $\bar{y}$, $\bar{z}$, and $\bar{t}$

H: Vertical distance scaling factor.

L: Horizontal distance scaling factor.

U: Horizontal velocity scaling factor.

λ: Aspect ratio; defined by $\frac{H}{L}$.

ϵ: Inverse Reynolds number; defined by $\frac{E}{UL}$.

C^2: Inverse Froude number; defined by $\frac{gH}{U^2}$.

K_{fr}: Coefficient of bottom surface friction.

K_{w}: Coefficient of upper surface friction; namely, wind friction.

u: Horizontal velocity when transformed to s-coordinates.

ϕ: A mathematical expression used to simplify the governing equations. Has the value $\phi = w - (a_x + sh_x)\, u - sh_t$. Can be used in post-processing to determine values for w.

$\bar{\bar{u}}$: Depth-averaged component of horizontal velocity; depth indepedent.

$\tilde{u}$: Residual component of horizontal velocity; depth dependent.

$\mathbf{u}$: Vector of horizontal velocities at all marker points in the fluid domain.

$\mathbf{u}_{\mathrm{w}}$: Vector of wind velocities at the surface at all fluid marker points.

$\mathbf{M}$: Mass matrix.

$\bar{\mathbf{M}}$: Viscosity matrix.

$\mathbf{M}_{\mathrm{w}}$, $\mathbf{M}_{\mathrm{fr}}$: Upper and lower surface frictions matrices.

1 Introduction

The United States Navy has enjoyed years of dominance in the open ocean. It is unchallenged as a blue-water power, but is facing increasing operational challenges in the littoral regions of the world. The majority of the world's population resides in coastal regions, which id also the scene of most of the world's conflict areas. Therefore, it is important for the Navy to expand its operational capability in shallow water regions such as estuaries and enclosed bays.

One significant way in which littoral regions differ from the open ocean is in the motion of the water there. In the open ocean, currents are largely predictable.The different seasons' prevailing currents have been charted since the days of the earliest oceanic explorers. However, currents in coastal regions are heavily affected by the shape of the local terrain—known as bathymetry—as well as tides. While in the open ocean the sea floor is of little consequence to all but the deepest-diving submarines, in coastal regions the sea floor affects ships of all sizes. Tidal influences in littoral regions affect navigation for both surface ships and submarines, as well as amphibious landings, special warfare, and non-combat operations such as preparation for storm surges.

To deal with these challenges more appropriately, the Navy would like to be able to simulate the motion of the sea in shallow water regions. To enable this the Navy needs a model that can predict the state of the water, or fluid, in the future based on the bathymetry and conditions in the fluid at the current time. The inputs to such a model would be the initial conditions of the state variables at the current time. The state variables are the unknown dependent variables of the problem; quantities such as velocity of fluid particles, height from the lower to the upper surface of the fluid, or water column, pressure, density, and more. The independent variables of this problem are location in three dimensional space, commonly expressed in Cartesian coordinates by the triple $\langle x, y, z \rangle$, and time. The output of this program will be the predicted values of these state variables at some time in the future.

This model is created using standard oceanographic equations as the starting point and develops these into the governing equations. The system of governing equations has two sources of non-linearity: the bathymetry and the Eulerian form of the governing equations. Because the problem is non-linear, a solution must be found through numerical methods.

2 The Princeton Ocean Model

Computer models already exist that predict future values of state variables from information available at the current time. One important model is the Princeton Ocean Model (POM). Much of the Navy's coastal ocean modeling software is derived from POM. The Princeton Ocean Model is a pioneering ocean modeling software developed by Alan Blumberg and George Mellor in 1977. POM is a numerical model that is not grid specific–it can be applied to any bathymetry. It is written in FORTRAN and has been extended and modified significantly over the past thirty years. The mathematical foundation of POM is a turbulence closure model developed by Mellor in 1973 and expanded in collaboration with Tetsuji Yamada over the next ten years. The Mellor-Yamada model is based on older turbulence closure hypotheses developed by Rotta [11] and Kolmogorov [5]. The governing equations of POM evaluate fluid properties, including velocity, temperature, and salinity, in a three dimensional space corresponding with Cartesian rectangular coordinates [10].

POM is developed from first principles of oceanography: conservation of mass, conservation of momentum, heat transport, and salt transport. These equations resemble the Navier-Stokes equations, except that they are relevant to turbulent rather than laminar flows. For shallow water estuary problems, all flow is assumed to be turbulent. POM replaces the Navier-Stokes stresses that are applicable to laminar flows with Reynolds stresses that are used for turbulent flow.

The model we propose in this project uses a similar mathematical approach to POM's leap-frog system for computing mean velocities. Both POM and our model first update the mean velocity at each point based on data from previous time steps, then update particle positions using the just calculated values of the mean velocity. After these steps, other quantities can be computed and then the cycle repeats by updating mean velocity at the next time step. This approach helps to preserve conserved quantities.

POM was not intended to address shallow water and estuary problems; rather, it was originally designed to solve for fluid motion on an oceanic scale and was later modified to deal with shallow water regions. It operates on horizontal scales of between 1 and 100 km, while the vertical scale is in tens to hundreds of meters. The time scale of POM ranges from tidal to monthly intervals [8].

2.1 Eulerian and Lagrangian Representations

The Eulerian and Lagrangian approaches are two different representations for fluid motion. In the Eulerian or field representation, the velocity and acceleration of a particle are represented in terms of that particle's position in space. In the Lagrangian representation, the velocity and acceleration are associated with specific particles. The Lagrangian is called a particle tracking representation because each individual particle has velocity and acceleration

functions assigned to it that follow that particle through all positions and times. In the Eulerian representation, velocity and acceleration are functions of the position in space at any given time [7].

Both POM and our model develop from the first principles of oceanography represented by Eulerian field equations because these field equations are simpler to develop than their Lagrangian counterparts. POM remains a primarily Eulerian model even in implementation. Since it is based on field equations, it does not do well with certain free boundary problems because the original model was not designed to track these boundaries. The free boundary is the physical interface between land and water. For a wet-dry problem, the model is asked to find the times when a surface is either wetted or dried because of tidal action, a storm surge, or other physical phenomena. The Eulerian field equations do not explicity track the motion of particles from a permanently wet area to an area that is sometimes dry. Instead, POM tries to calculate the velocity of the fluid at fixed grid points regardless of whether those points are in the wet or dry area. Furthermore, it has no mechanism to mark a "wet" area that has become "dry" or vice-versa [1].

Since POM has trouble distinguishing between wet and dry areas, and because we are interested in solving this sort of free-boundary problem, we take a slightly different approach. We start with the same equations, and develop them in a similar Eulerian method. However, we create a Lagrangian grid covering the lateral space of the fluid domain that moves with the flow of the water. The lateral space of the domain are the dimensions other than the vertical dimension. For example, when implementing a two-dimensional model from the two-dimensional shallow water equations, the lateral space is along the x-coordinate while the z-coordinate is the vertical space. With the three-dimensional model, however, the entire $x-y$ plane is the lateral space. The nodes in our grid are called fluid-markers and they are not fixed in space, nor are they fixed relative to other nearby fluid-markers. Instead, each fluid-marker point is associated with a particle and markers move as the velocity of the corresponding particle is re-calculated at each time step.

This method is better able to track the wet-dry boundary. By defining a certain set of fluid markers as the "edge" of the water, we can track where these particles move and thus track where the free boundary moves, allowing us to easily distinguish between wet and dry surfaces. We believe that for certain types of problems, especially those that deal with the shallow water estuaries in which we are interested, this free boundary tracking is a better way of accurately modeling coastal ocean circulation.

2.2 σ-Coordinate Transformation

The most important development of the Mellor model is the introduction of the σ-coordinate system which helps in dealing with significant topographical variability. The σ-transformation replaces the depth coordinate with σ, a number that ranges between -1 and

0. Each value of σ has a different z-coordinate at different points $\langle x, y \rangle$. This transformation reduces the vertical component of the domain of the problem to a regular rectangular shape in σ, and pushes the complexity added by irregular bathymetry into the governing equations. Since the depth dimension is divided into many "layers," each corresponding to a numerical value of σ, it is fundamentally different from the x and y directions [1].

The model we are developing uses a variation on the σ-coordinate transformation that we refer to as an s-coordinate transformation. Where σ ranges between 0 at the upper surface of the water and -1 at the lower surface, s varies from 1 at the upper surface to 0 at the bottom surface.

2.3 Numerical Viscosity

While POM has a complicated mechanism that uses the Reynolds Stress terms from the physical equations to remove high frequency oscillations from the model, we have chosen a simpler method. We approximate POM's eddy viscosity (ϵ) with a numerical viscosity. Because we make this approximation, we have the freedom to set the value of numerical viscosity and can therefore choose a value that will help cancel terms in our governing equations. We use a value of ϵ that is dependent on both the size of the time step and the size of the interval between marker points to help our numerical model to converge.

2.4 Asymptotic Solution to the Velocity Residual

POM associates a location in space represented by the point $\langle x, y \rangle$ with the averaged properties of the infinitesimally thin column of fluid above that point. The velocity at all depths above $\langle x, y \rangle$ is averaged into a mean velocity associated with that point. The difference from the mean at each vertical position in the water column is called the velocity residual. POM attempts to directly solve the vertical profile equation for this residual. However, we have made a significant contribution to work in this field by creating an asymptotic solution for the vertical profile. We show that the solution to the vertical profile can be split into transient and steady-state components and by obtaining appropriate estimates, demonstrate that the transient component can be ignored because it rapidly decays to zero. Therefore, the vertical profile can be quickly approximated with our model.

3 Physical Equations

The derivation of equations in this and the following sections borrows from many sources; including [2], [4], [6], [7], [9], [12], and [13].

Both our model and POM start with the basic equations of physical oceanography. The scale of the shallow water regions for which we design this model is such that the β-plane approximation can be adopted. Using the β-plane approximation, we model the curvature of the earth with a linear approximation and a Cartesian coordinate system; this simplifies the mathematics of our model. The cost of this assumption is that a linear factor must be added to account for the motion of the spherical earth. This is the Coriolis effect and will be discussed in greater detail later in this section.

We use both field and particle tracking equations as we develop and evaluate the governing equations of this model. When a particle is tracked in a Lagrangian representation, its location in space is described by $\langle X(\hat{t}), Y(\hat{t}), Z(\hat{t})\rangle$ while the velocity components associated with a point in space in the Eulerian representation is $\langle \hat{x}(\hat{t}), \hat{y}(\hat{t}), \hat{z}(\hat{t})\rangle$. The velocity of a particle in either representation is described by the vector $\mathbf{V}(\hat{x}, \hat{y}, \hat{z}, \hat{t})$ where

$$\mathbf{V} = \mathbf{i}\hat{u} + \mathbf{j}\hat{v} + \mathbf{k}\hat{w}$$

is such that $\hat{u}$, $\hat{v}$, and $\hat{w}$ represent the component of the velocity in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ directions, respectively. The identities for $\hat{u}$, $\hat{v}$, and $\hat{w}$ in Lagrangian notation are

$$\begin{aligned}
\frac{d}{d\hat{t}}X(\hat{t}) &= \hat{u}(X(\hat{t}), Y(\hat{t}), Z(\hat{t}), \hat{t}),\\
\frac{d}{d\hat{t}}Y(\hat{t}) &= \hat{v}(X(\hat{t}), Y(\hat{t}), Z(\hat{t}), \hat{t}),\\
\frac{d}{d\hat{t}}Z(\hat{t}) &= \hat{w}(X(\hat{t}), Y(\hat{t}), Z(\hat{t}), \hat{t}).
\end{aligned} \tag{1}$$

Using these notations, we derive our governing equations. In (Figure 1) we consider an elemental cube with its faces aligned to the coordinates $\langle \hat{x}, \hat{y}, \hat{z}\rangle$. If this cube is small enough that $\hat{u}$, $\hat{v}$, and $\hat{w}$ can be considered constant on all its faces, then the net volume flow through the $\hat{x}$ and $\hat{x} + \Delta\hat{x}$ faces is

$$(\hat{u} + \hat{u}_{\hat{x}}\Delta\hat{x})\,\Delta\hat{y}\Delta\hat{z} - \hat{u}\Delta\hat{y}\Delta\hat{z} = \hat{u}_{\hat{x}}\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}.$$

We use a similar derivation for the $\hat{y}$ and $\hat{z}$ faces and sum the net volume flow through all six faces to obtain

$$(\hat{u}_{\hat{x}} + \hat{v}_{\hat{y}} + \hat{w}_{\hat{z}})\,\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}.$$

The Boussinesq Approximation states that density differences between fluid elements can be ignored when approximating a flow, except where the density appears in a term multiplied by g, the acceleration due to gravity. This approximation applies in the case of the equation

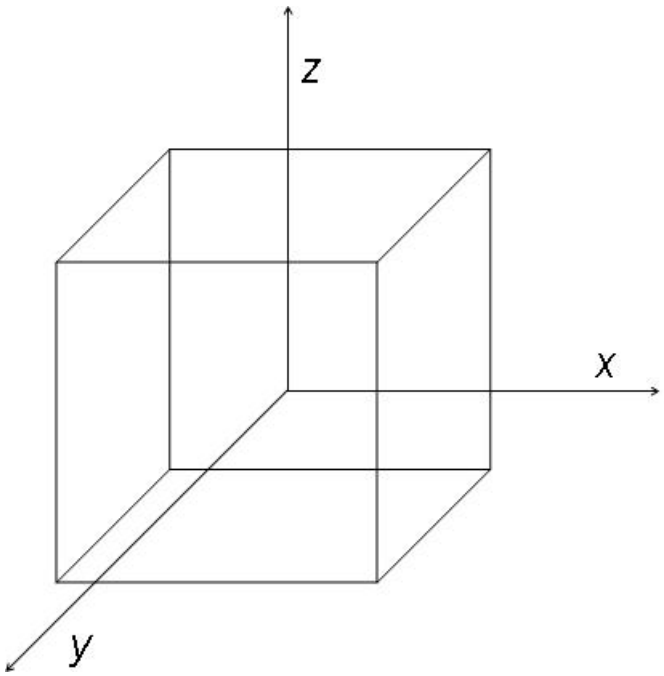

Figure 1: An elemental cube

above. Since we assume that fluid density is constant, we can assume that the mass inside the constant volume $\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}$ does not change. Therefore, we see that

$$\hat{u}_{\hat{x}} + \hat{v}_{\hat{y}} + \hat{w}_{\hat{z}} = 0 \tag{2}$$

or in vector notation

$$\nabla \cdot \mathbf{V} = 0.$$

This is the equation for mass continuity, or conservation of mass, which constitutes one of the four governing equations of our model.

The other three equations derive from Newton's Second Law of Motion. These momentum equations balance forces with accelerations. For the elemental cube in (Figure 2), the net force in the x-direction caused by pressure from the adjacent fluid is

$$-\hat{p}_{\hat{x}}\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}$$

while the net force due to viscous or turbulent stresses in that direction is

$$\left((\hat{\sigma}_{11})_{\hat{x}} + (\hat{\sigma}_{21})_{\hat{y}} + (\hat{\sigma}_{31})_{\hat{z}}\right)\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}.$$

The first subscript on the stress symbol $\hat{\sigma}$ signifies the coordinate normal to the face of the cube on which the stress acts, and the second subscript is the direction of the stress, where we replace $\hat{x}$ with 1, $\hat{y}$ with 2, and $\hat{z}$ with 3 to avoid confusion with the partial derivative

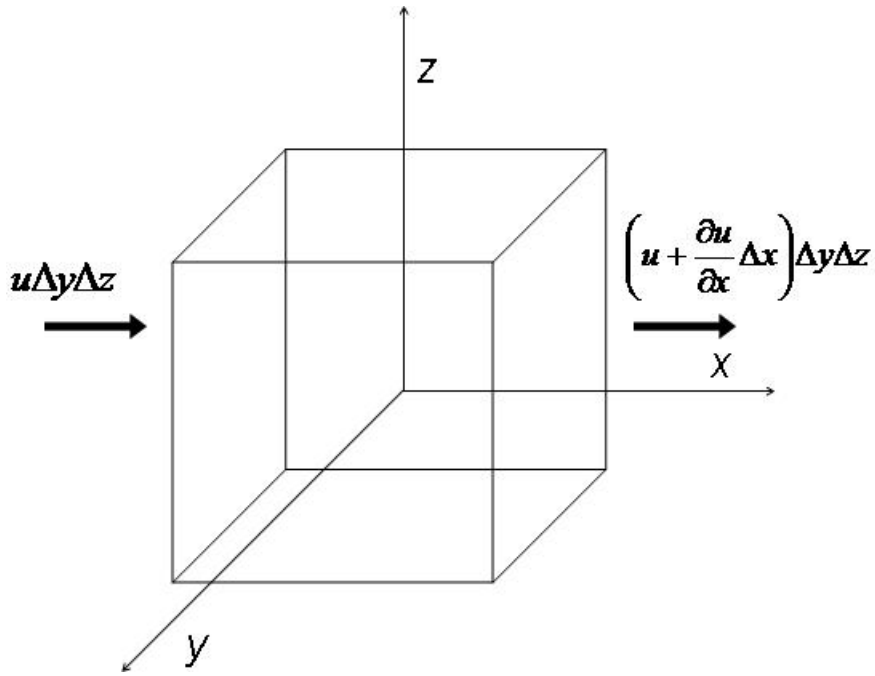

Figure 2: An elemental cube showing force in the x-direction

subscripts. These two forces are equated to the product of mass and acceleration according to Newton's Second Law. The mass of the elemental cube is

$$\rho\Delta\hat{x}\Delta\hat{y}\Delta\hat{z}$$

where ρ is the fluid density. Acceleration in the x-direction may be written as the total derivative of the velocity

$$\frac{D\hat{u}}{D\hat{t}} = \hat{u}\hat{u}_{\hat{x}} + \hat{v}\hat{u}_{\hat{y}} + \hat{w}\hat{u}_{\hat{z}} + \hat{u}_{\hat{t}}.$$

However, this value of the acceleration is only useful for a body of water in "absolute" coordinates. Our fluid exists in a rotating system on the surface of the earth, so a correction factor must be added to convert to a "relative" coordinate system. This correction factor is called the Coriolis acceleration and changes the net acceleration to

$$\hat{u}\hat{u}_{\hat{x}} + \hat{v}\hat{u}_{\hat{y}} + \hat{w}\hat{u}_{\hat{z}} + \hat{u}_{\hat{t}} - f\hat{v} + f_{\hat{y}}\hat{w}$$

where f is the Coriolis parameter that is equal to two times the angular velocity of the earth times the sine of the latitude. The second Coriolis term is generally taken to be small for oceanographic models and we ignore it here. Equating the force terms with the product of the mass and acceleration terms yields the conservation of momentum equation in the x-direction, which is

$$\rho\left(\hat{u}\hat{u}_{\hat{x}} + \hat{v}\hat{u}_{\hat{y}} + \hat{w}\hat{u}_{\hat{z}} + \hat{u}_{\hat{t}} - f\hat{v}\right) = -\hat{p}_{\hat{x}} + (\hat{\sigma}_{11})_{\hat{x}} + (\hat{\sigma}_{21})_{\hat{y}} + (\hat{\sigma}_{31})_{\hat{z}} \tag{3}$$

A similar derivation yields the y- and z-momentum conservation laws. These equations have different terms for the Coriolis acceleration and the z-momentum equation has a term for the hydrostatic effects of gravity. These equations are

$$\rho\left(\hat{u}\hat{v}_{\hat{x}} + \hat{v}\hat{v}_{\hat{y}} + \hat{w}\hat{v}_{\hat{z}} + \hat{v}_{\hat{t}} + f\hat{u}\right) = -\hat{p}_{\hat{y}} + (\hat{\sigma}_{12})_{\hat{x}} + (\hat{\sigma}_{22})_{\hat{y}} + (\hat{\sigma}_{32})_{\hat{z}} \tag{4}$$

$$\rho\left(\hat{u}\hat{w}_{\hat{x}} + \hat{v}\hat{w}_{\hat{y}} + \hat{w}\hat{w}_{\hat{z}} + \hat{w}_{\hat{t}} - f_{\hat{y}}\hat{u}\right) = -\rho g - \hat{p}_{\hat{z}} + (\hat{\sigma}_{13})_{\hat{x}} + (\hat{\sigma}_{23})_{\hat{y}} + (\hat{\sigma}_{33})_{\hat{z}} \tag{5}$$

where g is the acceleration due to gravity.

3.1 Governing Equations for the Two-Dimensional Model

Equations (2)-(5) are the governing equations of our model. We have omitted all factors relating to temperature and salinity, two other properties that must be conserved in addition to mass and momentum. We also take density to be constant. For simplicity, we take $\rho = 1$ with no loss in generality; we do this by multiplying through the stresses and pressure by $\frac{1}{\rho}$.

In addition to the assumptions of the previous section, we will make another simplifying change. We first consider a set of governing equations with reduced complexity. We choose to ignore the y-direction and focus instead only on movement in two dimensions. Our goal is to analyze the impact of the horizontal coordinates relative to the depth coordinates. Assuming that all quantities are independent of the y-coordinate reduces our problem from a sloshing tank of fluid to a thin fluid sheet trapped between two panes on either side. In this context, the Coriolis acceleration has no meaning; the Coriolis terms come out of the governing equations when we remove all terms dependent on the y-coordinate. The four governing equations are reduced to three: mass continuity, conservation of momentum in the $\hat{x}$ direction, and conservation of momentum in the $\hat{z}$ direction;

$$\hat{u}_{\hat{x}} + \hat{w}_{\hat{z}} = 0, \tag{6}$$

$$\hat{u}_{\hat{t}} + \hat{u}\hat{u}_{\hat{x}} + \hat{w}\hat{u}_{\hat{z}} + \hat{p}_{\hat{x}} = (\hat{\sigma}_{11})_{\hat{x}} + (\hat{\sigma}_{13})_{\hat{z}}\,, \tag{7}$$

$$\hat{w}_{\hat{t}} + \hat{u}\hat{w}_{\hat{x}} + \hat{w}\hat{w}_{\hat{z}} + \hat{p}_{\hat{z}} = (\hat{\sigma}_{31})_{\hat{x}} + (\hat{\sigma}_{33})_{\hat{z}} - g, \tag{8}$$

where g is the acceleration of gravity.

We assume the following constitutive relations among stresses and strains

$$\hat{\sigma}_{11} = 2E\hat{u}_{\hat{x}}, \quad \hat{\sigma}_{33} = 2E\hat{w}_{\hat{z}}, \quad \hat{\sigma}_{13} = \hat{\sigma}_{31} = E(\hat{u}_{\hat{z}} + \hat{w}_{\hat{x}}). \tag{9}$$

These relationships follow the form of the *Navier-Stokes* equations. However, the Eddy Viscosity (E) is substituted for the kinematic viscosity (ν) because kinematic viscosity applies to laminar flow and is not appropriate for the turbulent flow of coastal ocean circulation.

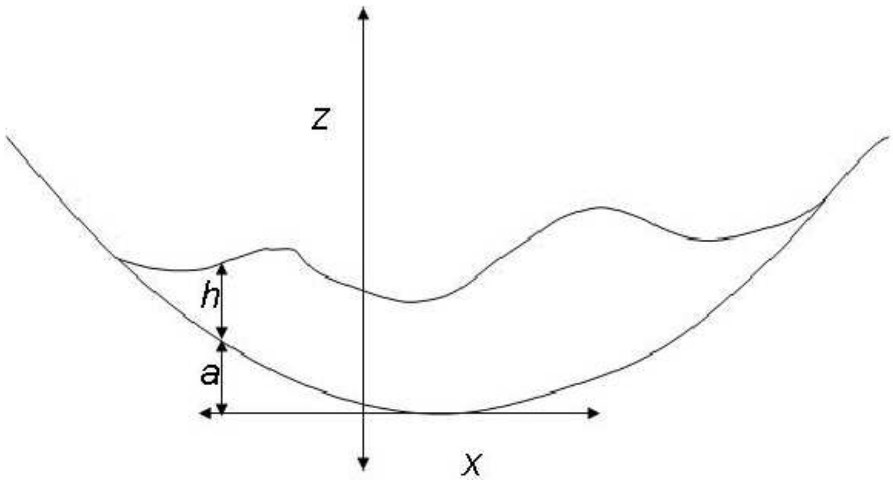

Figure 3: The domain of the two-dimensional model

3.2 Boundary Conditions

There are two surfaces which form the boundaries of the fluid in question. The lower surface is given by $\hat{z} = \hat{a}(\hat{x})$ and the upper surface by $\hat{z} = \hat{a}(\hat{x}) + \hat{h}(\hat{x}, \hat{t})$ where $\hat{h}(\hat{x}, \hat{t})$ is the height of the water column as seen in (Figure 3). We assume that fluid particles located on these surfaces at one time remain there for all times. The mathematical expression of this implication for the bottom surface is

$$\frac{d}{d\hat{t}}\left(Z(\hat{t}) - \hat{a}(X(\hat{t}))\right) = 0$$

using our Lagrangian notation. Noting the definitions in (1), we use the chain rule to differentiate the previous equation by t and express this boundary condition as

$$\hat{w}(\hat{x}, \hat{a}, \hat{t}) - \hat{a}_{\hat{x}}\hat{u}(\hat{x}, \hat{a}, \hat{t}) = 0. \tag{10}$$

The complementary equation for the upper surface is

$$\begin{aligned} 0 &= \frac{d}{d\hat{t}}\left(Z(\hat{t}) - \left(\hat{a}(X(\hat{t})) + \hat{h}(X(\hat{t}), \hat{t})\right)\right) \\ &= \hat{w}(\hat{x}, \hat{a} + \hat{h}, \hat{t}) - \hat{a}_{\hat{x}}\hat{u}(\hat{x}, \hat{a} + \hat{h}, \hat{t}) - \hat{h}_{\hat{x}}\hat{u}(\hat{x}, \hat{a} + \hat{h}, \hat{t}) - \hat{h}_{\hat{t}}. \end{aligned} \tag{11}$$

An additional boundary condition is that pressure at the free surface is the same when measured from either side of the boundary. We assume that the pressure of the atmosphere above the free surface is zero. Thus, the pressure at the free surface is also zero. This is expressed as

$$\hat{p}(\hat{x}, \hat{a} + \hat{h}, \hat{t}) = 0. \tag{12}$$

Boundary conditions that equate the turbulent stresses on the upper and lower surfaces to the frictional forces and wind forces on these surfaces are also required. We defer derivation of these conditions until we transform the physical equations into their non-dimensional forms because the introduction of scaling factors will greatly simplify that derivation.

Lastly, we note that initial conditions are required for $\hat{u}$, $\hat{w}$, and $\hat{h}$ in order to solve this system of equations.

Summary of Equations

Mass Continuity	$\hat{u}_{\hat{x}} + \hat{w}_{\hat{z}} = 0$	(6)
x-Momentum	$\hat{u}_{\hat{t}} + \hat{u}\hat{u}_{\hat{x}} + \hat{w}\hat{u}_{\hat{z}} + \hat{p}_{\hat{x}} = (\hat{\sigma}_{11})_{\hat{x}} + (\hat{\sigma}_{13})_{\hat{x}}$	(7)
z-Momentum	$\hat{w}_{\hat{t}} + \hat{u}\hat{w}_{\hat{x}} + \hat{w}\hat{w}_{\hat{z}} + \hat{p}_{\hat{z}} = (\hat{\sigma}_{13})_{\hat{x}} + (\hat{\sigma}_{33})_{\hat{z}} - g$	(8)
Lower Surface Boundary	$\hat{w}(\hat{a}) - \hat{a}_{\hat{x}}\hat{u}(\hat{a}) = 0$	(10)
Upper Surface Boundary	$\hat{w}(\hat{x}, \hat{a}+\hat{h}, \hat{t}) - \left(\hat{a}+\hat{h}\right)_{\hat{x}} \hat{u}(\hat{x}, \hat{a}+\hat{h}, \hat{t}) - \hat{h}_{\hat{t}} = 0$	(11)

4 Shallow Water Scaling

We non-dimensionalize the variables in our equation by assigning a scaling factor to all dependent and independent variables. The independent scaling factors are

$$\hat{x} = L\bar{x}, \quad \hat{z} = H\bar{z}, \quad \hat{t} = \tfrac{L}{U}\bar{t}, \tag{13}$$

while the dependent scaling factors are

$$\begin{aligned} \hat{h}(\hat{x},\hat{t}) &= H\bar{h}(\bar{x},\bar{t}), & \hat{a}(\hat{x}) &= H\bar{a}(\bar{x}), \\ \hat{u}(\hat{x},\hat{z},\hat{t}) &= U\bar{u}(\bar{x},\bar{z},\bar{t}), & \hat{w}(\hat{x},\hat{z},\hat{t}) &= \tfrac{UH}{L}\bar{w}(\bar{x},\bar{z},\bar{t}), \\ \hat{p}(\hat{x},\hat{z},\hat{t}) &= gH\bar{p}(\bar{x},\bar{z},\bar{t}). \end{aligned} \tag{14}$$

The scaling factor L is applied to horizontal coordinates while the scaling factor H is applied to vertical coordinates. U represents a typical horizontal velocity. These two scaling factors approximate the dimensions of the model's domain. Since the hydrostatic pressure at a point depends on the weight of the water column above that point, the pressure is scaled by the product gH where g is the acceleration due to gravity. From these basic scaling factors we can see that $\bar{x}$ and $\bar{u}$ are related by the identity $\bar{u} = \bar{x}_{\bar{t}}$.

Following the convention of other works in this area [1], we define three dimensionless scalars λ, ϵ, and C^2 as follows

$$\lambda = \tfrac{H}{L}, \quad \epsilon = \tfrac{E}{UL}, \quad C^2 = \tfrac{gH}{U^2}. \tag{15}$$

These scalars are called the aspect ratio (λ), the inverse Reynolds number (ϵ), and the inverse Froude number (C^2). They appear as coefficients in our dimensionless equations. In most shallow water situations, the vertical dimension will be much smaller than the horizontal dimension. In the shallow water estuary problems, water depth is seldom over 50 meters but horizontal distances are hundreds of kilometers. We therefore make the following assumptions about the relationships among these scaling factors:

$$0 < \lambda \ll 1, \quad \epsilon \ll 1, \quad C^2 \gg 1. \tag{16}$$

The relationships between these scaling factors will be important for justifying mathematical simplifications of our governing equations.

The scalings for the σ terms follow directly from (13) and (14). These scalings are

$$\begin{aligned} \hat{\sigma}_{11} &= 2E\hat{u}_{\hat{x}} = 2\frac{EU}{L}\bar{u}_{\bar{x}}, \\ \hat{\sigma}_{33} &= 2E\hat{w}_{\hat{z}} = 2\frac{EU}{L}\bar{w}_{\bar{z}}, \end{aligned} \tag{17}$$

$$\hat{\sigma}_{13} = \hat{\sigma}_{31} = E\left(\hat{u}_{\hat{z}} + \hat{w}_{\hat{x}}\right) = \frac{EU}{L}\left(\frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}}\right).$$

With scaling factors defined for all variables, the equations of motion and boundary conditions can be translated into dimensionless form.

4.1 Dimensionless Mass Continuity Equation

We non-dimensionalize the mass continuity equation (6) using the identities in (14) to obtain

$$\begin{aligned} 0 &= \hat{u}_{\hat{x}} + \hat{w}_{\hat{z}} \\ &= U\bar{u}_{\bar{x}}\bar{x}_{\hat{x}} + \frac{UH}{L}\bar{w}_{\bar{z}}\bar{z}_{\hat{z}} \end{aligned}$$

We evaluate $\bar{x}_{\hat{x}}$ and $\bar{z}_{\hat{z}}$ using (13) and reduce the above equation to

$$\frac{U}{L}\bar{u}_{\bar{x}} + \frac{U}{L}\bar{w}_{\bar{z}} = 0,$$

or,

$$\bar{u}_{\bar{x}} + \bar{w}_{\bar{z}} = 0. \tag{18}$$

4.2 Dimensionless z-Momentum Equation and the Hydrostatic Approximation

Using the scaling factors from (13)-(17), the z-momentum equation (8) becomes

$$\frac{U^2H}{L^2}\bar{w}_{\bar{t}} + \frac{U^2H}{L^2}\left(\bar{u}\bar{w}\right)_{\bar{x}} + \frac{U^2H}{L^2}\left(\bar{w}^2\right)_{\bar{z}} + g\bar{p}_{\bar{z}} = \frac{EU}{L^2}\left(\frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}}\right)_{\bar{x}} + 2\frac{EU}{HL}\left(\bar{w}_{\bar{z}}\right)_{\bar{z}} - g$$

which can be written as

$$\lambda^2\left[\bar{w}_{\bar{t}} + \left(\bar{u}\bar{w}\right)_{\bar{x}} + \left(\bar{w}^2\right)_{\bar{z}}\right] + C^2\left(\bar{p}_{\bar{z}} + 1\right) = \epsilon\left[\left(\bar{u}_{\bar{z}} + \lambda^2\bar{w}_{\bar{x}}\right)_{\bar{x}} + 2\left(\bar{w}_{\bar{z}}\right)_{\bar{z}}\right] \tag{19}$$

Until this point our model has been exact; no terms have been dropped to simplify the problem. Now with the z-momentum equation (19) and the scalars in (15), we have the opportunity to do so. In the z-momentum equation all terms are multiplied by one of the factors λ, ϵ, or C^2. Recalling the relationships between scaling factors from (16)—that $0 < \lambda \ll 1$, $0 < \epsilon \ll 1$, and $C^2 \gg 1$—we see that the other terms of the z-momentum equation are insignificant compared to the term multiplied by C^2. Removing these, our approximation is

$$C^2(\bar{p}_{\bar{z}} + 1) = 0.$$

Now we integrate this last identity with respect to the vertical coordinate from an arbitrary point $\bar{z}$ to the free surface of the fluid at $\bar{a} + \bar{h}$ to get

$$\int_{\bar{z}}^{\bar{a}+\bar{h}} \left(\bar{p}_{\bar{z}} + 1\right)\, d\eta = \bar{p}(\bar{x}, \bar{a} + \bar{h}, \bar{t}) - \bar{p}(\bar{x}, \bar{z}, \bar{t}) + (\bar{a} + \bar{h}) - \bar{z} = 0.$$

Since the pressure at the free surface is zero, we obtain

$$\bar{p} = \bar{a} + \bar{h} - \bar{z}. \tag{20}$$

This last identity is referred to as the hydrostatic approximation.

4.3 Dimensionless x-Momentum Equation

The x-momentum equation non-dimensionalizes similarly. From (7) we have

$$\hat{u}_{\hat{t}} + \left(\hat{u}^2\right)_{\hat{x}} + \left(\hat{w}\hat{u}\right)_{\hat{z}} + \hat{p}_{\hat{x}} = \left(\hat{\sigma}_{11}\right)_{\hat{x}} + \left(\hat{\sigma}_{13}\right)_{\hat{x}}$$

We replace the scaled parameters with the non-dimensional versions to get

$$\frac{U^2}{L}\bar{u}_{\bar{t}} + \frac{U^2}{L}\left(\bar{u}^2\right)_{\bar{x}} + \frac{U^2}{L}\left(\bar{w}\bar{u}\right)_{\bar{z}} + \frac{gH}{L}\bar{p}_{\bar{x}} = 2\left(\frac{EU}{L^2}\bar{u}_{\bar{x}}\right)_{\bar{x}} + \left(\frac{EU}{HL}\left(\frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}}\right)\right)_{\bar{z}}$$

or

$$\bar{u}_{\bar{t}} + \left(\bar{u}^2\right)_{\bar{x}} + \left(\bar{w}\bar{u}\right)_{\bar{z}} + C^2\bar{p}_{\bar{x}} = 2\left(\epsilon\bar{u}_{\bar{x}}\right)_{\bar{x}} + \left(\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}}\right)_{\bar{z}} + \left(\epsilon\bar{w}_{\bar{x}}\right)_{\bar{z}}$$

with C^2, ϵ, and λ as defined in (16).

In the situation where ϵ is constant, we exploit the identity $\bar{w}_{\bar{z}} = -\bar{u}_{\bar{x}}$ (18) to reduce the right hand side of the last equation. Also, we use the hydrostatic approximation (20) to replace $\bar{p}_{\bar{x}}$ with $\bar{a}_{\bar{x}} + \bar{h}_{\bar{x}}$ to obtain

$$\bar{u}_{\bar{t}} + \left(\bar{u}^2\right)_{\bar{x}} + \left(\bar{w}\bar{u}\right)_{\bar{z}} + C^2\left(\bar{a}_{\bar{x}} + \bar{h}_{\bar{x}}\right) = \left(\epsilon\bar{u}_{\bar{x}}\right)_{\bar{x}} + \left(\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}}\right)_{\bar{z}} \tag{21}$$

4.4 Dimensionless Boundary Conditions

The boundary condition at the bottom surface (10) and the free surface (11) transform to

$$\bar{w}\left(\bar{x}, \bar{a}(\bar{x})\right) = \bar{u}\bar{a}_{\bar{x}} \tag{22}$$

and

$$\bar{w}\left(\bar{x}, \bar{a}(\bar{x}) + \bar{h}(\bar{x}, \bar{t}), \bar{t}\right) = \bar{h}_{\bar{t}} + \bar{u}(\bar{a} + \bar{h})_{\bar{x}}. \tag{23}$$

4.5 Dimensionless Surface Traction Approximation

The derivation of the surface traction approximations in (A) is taken from [4]. The results of this derivation are two boundary conditions, (A-8) and (A-9). These can be written as

$$\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}} - \epsilon\bar{a}_{\bar{x}}\bar{u}_{\bar{x}} = \frac{\epsilon}{\lambda^2}K_{\text{fr}}\,\bar{h}\,\bar{u}(\bar{x}, \bar{a}, \bar{t}) \tag{24}$$

for the lower surface $\bar{z} = \bar{a}$ and

$$\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}} - \epsilon\left(\bar{a}_{\bar{x}} + \bar{h}_{\bar{x}}\right)\bar{u}_{\bar{x}} = \frac{\epsilon}{\lambda^2}K_{\mathrm{w}}\left(u_{\mathrm{w}} - \bar{u}(\bar{x}, \bar{a} + \bar{h}, \bar{t})\right) \tag{25}$$

for the upper surface where $\bar{z} = \bar{a} + \bar{h}$. K_{w} is the coefficient of friction between the fluid and the air at the free surface, K_{fr} is the coefficient of friction between the fluid and the bottom surface, and $u_{\mathrm{w}}(x, t)$ is the velocity of the wind at the free surface.

Summary of Equations

Mass Continuity	$\bar{u}_{\bar{x}} + \bar{w}_{\bar{z}} = 0$	(18)
x-Momentum	$\bar{u}_{\bar{t}} + (\bar{u}^2)_{\bar{x}} + (\bar{w}\bar{u})_{\bar{z}} + C^2\left(\bar{a}_{\bar{x}} + \bar{h}_{\bar{x}}\right) = (\epsilon\bar{u}_{\bar{x}})_{\bar{x}} + \left(\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}}\right)_{\bar{z}}$	(21)
Upper Surface Boundary	$\bar{w}\left(\bar{x}, \bar{a}, \bar{t}\right) = \bar{u}\bar{a}_{\bar{x}}$	(22)
Lower Surface Boundary	$\bar{w}\left(\bar{x}, \bar{a} + \bar{h}, \bar{t}\right) = \bar{h}_{\bar{t}} + \bar{u}(\bar{a} + \bar{h})_{\bar{x}}$	(23)
Lower Boundary Traction	$\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}} - \epsilon\bar{a}_{\bar{x}}\bar{u}_{\bar{x}} = \frac{\epsilon}{\lambda^2}K_{\mathrm{fr}}\,\bar{h}\,\bar{u}(\bar{x}, \bar{a}, \bar{t})$	(24)
Upper Boundary Traction	$\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}} - \epsilon\left(\bar{a}_{\bar{x}} + \bar{h}_{\bar{x}}\right)\bar{u}_{\bar{x}} = \frac{\epsilon}{\lambda^2}K_{\mathrm{w}}\left(u_{\mathrm{w}} - \bar{u}(\bar{x}, \bar{a} + \bar{h}, \bar{t})\right)$	(25)

5 s-Coordinate Transformation

We now introduce this fundamental transformation to map our equations of motion from a variable domain in the vertical direction to a fixed domain in the vertical direction. This simplifies our bathymetry and moves complexity from the domain to the governing equations by replacing a formerly irregular domain with a regular shape with constant height 1. We let

$$x = \bar{x}, \quad t = \bar{t}, \quad s = \frac{\bar{z} - \bar{a}(\bar{x})}{\bar{h}(\bar{x}, \bar{t})}, \quad a(x) = \bar{a}(\bar{x}), \quad h(t, x) = \bar{h}(\bar{x}, \bar{t}). \tag{26}$$

Then

$$\partial_{\bar{x}} = \partial_x - \frac{a_x + sh_x}{h}\partial_s, \quad \partial_{\bar{z}} = \frac{1}{h}\partial_s, \quad \partial_{\bar{t}} = \partial_t - \frac{sh_t}{h}\partial_s, \tag{27}$$

and

$$u(x, s, t) = \bar{u}(\bar{x}, \bar{z}, \bar{t}), \quad w(x, s, t) = \bar{w}(\bar{x}, \bar{z}, \bar{t}). \tag{28}$$

For the free boundary of the fluid domain—the interface between wet and dry areas where $h = 0$—division by h produces a singularity. This problem will be addressed in later chapters as we numerically approximate the governing equations. Note that on the bottom surface $\bar{z} = \bar{a}(\bar{x})$ and

$$s = \frac{\bar{a}(\bar{x}) - \bar{a}(\bar{x})}{\bar{h}(\bar{t}, \bar{x})} = 0,$$

whereas on the free surface $\bar{z} = \bar{a}(\bar{x}) + \bar{h}(\bar{x}, \bar{t})$ and

$$s = \frac{\big(\bar{a}(\bar{x}) + \bar{h}(\bar{x}, \bar{t})\big) - \bar{a}(\bar{x})}{\bar{h}(\bar{x}, \bar{t})} = 1.$$

We have changed the physical domain from a region defined by $\bar{a}(\bar{x}) < \bar{z} < \bar{a}(\bar{x}) + \bar{h}(\bar{x}, \bar{t})$ to a vertical domain defined by $0 < s < 1$. This simplification comes at the cost of adding complexity to the governing equations.

Following the precedent of previous works [1], we introduce the function ϕ, defined by

$$\phi(x, s, t) = w - (a_x + sh_x)u - sh_t$$

or

$$w = \phi + (a_x + sh_x)u + sh_t. \tag{29}$$

The function ϕ has no simple physical interpretation. Instead, it is a mathematical artifact that exists to simplify the governing equations by removing w. In both the mass continuity and x-momentum equations, all terms containing w can be replaced with an equivalent term containing ϕ. An important property for simplification is that ϕ satisfies the boundary conditions

$$\phi(x, 0, t) = \phi(x, 1, t) = 0 \tag{30}$$

which can be seen by putting (22) and (23) into (29).

5.1 Mass Continuity Equation in s-Coordinates

The mass continuity equation (18) transforms into s-coordinates as follows:

$$\begin{aligned} 0 &= \bar{u}_{\bar{x}} + \bar{w}_{\bar{z}} \\ &= u_x - \left(\frac{a_x + sh_x}{h}\right) u_s + \frac{1}{h} w_s \end{aligned}$$

If we multiply the last equation through by h we obtain

$$0 = hu_x - (a_x + sh_x)u_s + w_s, \tag{31}$$

Differentiating the function ϕ defined in (29) with respect to s yields

$$w_s = \phi_s + h_x u + (a_x + sh_x)u_s + h_t. \tag{32}$$

If we replace w_s in (31) by (32), we see that the continuity equation reduces to

$$h_t + (hu)_x + \phi_s = 0. \tag{33}$$

This equation is exact.

5.2 x-Momentum Equation in s-Coordinates

When the x-momentum equation (21) is expressed in s-coordinates and multiplied through by h, the left side of this equation becomes

$$\begin{aligned} (hu_t - sh_t u_s) + (huu_x - (a_x + sh_x)uu_s + huu_x - (a_x + sh_x)uu_s) \\ + (wu_s + w_s u) + C^2 h\,(a_x + h_x) = ... \end{aligned}$$

Noting that a_s and h_s are both zero allows us to remove those terms. We now use w and w_s as defined in (29) and (32) to simplify the left hand side to

$$(hu)_t + (hu^2)_x + (\phi u)_s + C^2 h\,(a_x + h_x) = ...$$

Applying the definitions in (26)-(28), the right hand side of the x-momentum equation (21) transforms to

$$\begin{aligned} ... &= (\epsilon \bar{u}_{\bar{x}})_{\bar{x}} + \left(\frac{\epsilon}{\lambda^2}\bar{u}_{\bar{z}}\right)_{\bar{z}} \\ ... &= \left(\epsilon u_x - \epsilon \frac{a_x + sh_x}{h} u_s\right)_x - \frac{a_x + sh_x}{h}\left(\epsilon u_x - \epsilon \frac{a_x + sh_x}{h} u_s\right)_s + \left(\frac{\epsilon}{\lambda^2}\frac{1}{h}\left(\frac{1}{h}u_s\right)\right)_s. \end{aligned}$$

If we multiply this expression through by h and note that we can move h in or out of an s derivative because h is independent of s, we find that the right hand side reduces to

$$... = \left(\frac{\epsilon}{\lambda^2 h} u_s\right)_s + h\left(\epsilon\left(u_x - \frac{a_x + sh_x}{h} u_s\right)\right)_x - \epsilon\,(a_x + sh_x)\left(u_x - \frac{a_x + sh_x}{h} u_s\right)_s.$$

Next we transform the right hand side of this equation into conservation form to obtain

$$... = \left(\frac{\epsilon}{\lambda^2 h}\tilde{u}_s\right)_s + \left(h\epsilon\left(u_x - \frac{a_x + sh_x}{h}u_s\right)\right)_x - \left(\epsilon\left(a_x + sh_x\right)\left(u_x - \frac{a_x + sh_x}{h}u_s\right)\right)_s .$$

The final scaled version of the x-Momentum equation is

$$\begin{aligned}(hu)_t + (hu^2)_x + (\phi u)_s + C^2 h\left(a_x + h_x\right) = \\ \left(\frac{\epsilon}{\lambda^2 h}u_s\right)_s + \left(h\epsilon\left(u_x - \frac{a_x + sh_x}{h}u_s\right)\right)_x - \left(\epsilon\left(a_x + sh_x\right)\left(u_x - \frac{a_x + sh_x}{h}u_s\right)\right)_s .\end{aligned} \quad (34)$$

5.3 Surface Traction Boundary Conditions in s-Coordinates

The equations (24) and (25) transform to

$$\frac{\epsilon}{\lambda^2 h}u_s(0) - \epsilon a_x\left(u_x(0) - \frac{a_x}{h}u_s(0)\right) = \frac{\epsilon}{\lambda^2}hK_{\text{fr}}u(0) \quad (35)$$

and

$$\frac{\epsilon}{\lambda^2 h}u_s(1) - \epsilon\left(a_x + h_x\right)\left(u_x(1) - \frac{a_x + h_x}{h}u_s(1)\right) = \frac{\epsilon}{\lambda^2}K_{\text{w}}\left(u_{\text{w}} - u(1)\right). \quad (36)$$

Summary of Equations

Mass Continuity

$$h_t + (hu)_x + \phi_s = 0 \quad (33)$$

x-Momentum

$$\begin{aligned}&(hu)_t + (hu^2)_x + (\phi u)_s + C^2 h\left(a_x + h_x\right) = \\ &\left(\tfrac{\epsilon}{\lambda^2 h}u_s\right)_s + \left(h\epsilon\left(u_x - \tfrac{a_x + sh_x}{h}u_s\right)\right)_x - \left(\epsilon\left(a_x + sh_x\right)\left(u_x - \tfrac{a_x + sh_x}{h}u_s\right)\right)_s\end{aligned} \quad (34)$$

Lower Boundary Traction

$$\tfrac{\epsilon}{\lambda^2 h}u_s(0) - \epsilon a_x\left(u_x(0) - \tfrac{a_x}{h}u_s(0)\right) = \tfrac{\epsilon}{\lambda^2}hK_{\text{fr}}u(0) \quad (35)$$

Upper Boundary Traction

$$\tfrac{\epsilon}{\lambda^2 h}u_s(1) - \epsilon\left(a_x + h_x\right)\left(u_x(1) - \tfrac{a_x + h_x}{h}u_s(1)\right) = \tfrac{\epsilon}{\lambda^2}K_{\text{w}}\left(u_{\text{w}} - u(1)\right) \quad (36)$$

6 Separating Mean and Residual Velocities

The variable $u(x, s, t)$ represents the lateral velocity of the fluid in the artifical domain created by the s-transformation. The x-direction is the only direction of free motion in this domain, while in the physical domain there is motion in both the x- and z-directions. The horizontal velocity varies over the water column of the fluid; this is expressed mathematically through the dependence of u on s. We split u into two components: one that is depth-dependent, and one that is independent of s. This decomposition is expressed by

$$u(x, s, t) = \bar{\bar{u}}(x, t) + \tilde{u}(x, s, t) \tag{37}$$

where

$$\bar{\bar{u}}(x, t) = \int_0^1 u(x, s, t)\, ds,$$
$$\tilde{u}(x, s, t) = u(x, s, t) - \bar{\bar{u}}(x, t).$$

We refer to $\bar{\bar{u}}$ as the depth-averaged horizontal velocity and $\tilde{u}$ is the horizontal velocity residual. Additionally, $\tilde{u}$ satisfies

$$\int_0^1 \tilde{u}(x, s, t)\, ds = 0. \tag{38}$$

6.1 Depth-Averaged Mass Continuity Equation

When we apply the decomposition (37) for u to the mass continuity equation (33), the result is

$$h_t + (h(\bar{\bar{u}} + \tilde{u}))_x + \phi_s = 0.$$

If we integrate this equation over s from 0 to 1 and exploit the boundary conditions in (30) and the identity (38), we obtain

$$h_t + (h\bar{\bar{u}})_x = 0 \tag{39}$$

With this transformation, the mathematical artifact ϕ is removed from the mass continuity equation. This was the purpose of replacing terms dependent on $\bar{w}$ with ϕ. While calculation of values of $\bar{w}$ is not important to determining horizontal velocity fields, we can still use information about ϕ to determine values for $\bar{w}$ in post-processing. If we subtract (39) from (33) we see that

$$\phi_s = -(h\tilde{u})_x.$$

We then use $\phi(x, 0, t) = 0$, one of the boundary conditions of ϕ in (30), to obtain

$$\phi(x, s, t) = -\left(h \int_0^s \tilde{u}(x, \eta, t)\, d\eta\right)_x. \tag{40}$$

Because of (38), the function ϕ given in (40) satisfies $\phi(x, 1, t) = 0$, the other boundary condition in (30). We now have an expression that we can evaluate to solve for ϕ and by extension $\bar{w}$. Although results for these values are not calculated in this project, that calculation would be a useful extension of this work.

6.2 Depth-Averaged x-Momentum Equation

We apply the decomposition in (37) to the x-momentum equation (34) to obtain

$$(h(\bar{\bar{u}}+\tilde{u}))_t + \left(h(\bar{\bar{u}}^2 + 2\bar{\bar{u}}\tilde{u} + \tilde{u}^2)\right)_x + (\phi(\bar{\bar{u}}+\tilde{u}))_s + C^2 h(a_x + h_x) = \left(\frac{\epsilon}{\lambda^2 h}(\bar{\bar{u}}+\tilde{u})_s\right)_s + \left(h\epsilon\left((\bar{\bar{u}}+\tilde{u})_x - \frac{a_x + sh_x}{h}(\bar{\bar{u}}+\tilde{u})_s\right)\right)_x - \left(\epsilon(a_x+sh_x)(\bar{\bar{u}}+\tilde{u})_x - \frac{a_x+sh_x}{h}(\bar{\bar{u}}+\tilde{u})_s\right)_s.$$

Separating terms and noting that $\bar{\bar{u}}_s = 0$ yields

$$(h\bar{\bar{u}})_t + (h\tilde{u})_t + (h\bar{\bar{u}}^2)_x + 2(h\bar{\bar{u}}\tilde{u})_x + (h\tilde{u}^2)_x + \phi_s\bar{\bar{u}} + (\phi\tilde{u})_s + C^2h(a_x+h_x) = \left(\frac{\epsilon}{\lambda^2 h}\tilde{u}_s\right)_s + \left(h\epsilon\left((\bar{\bar{u}}+\tilde{u})_x - \frac{a_x+sh_x}{h}\tilde{u}_s\right)\right)_x - \left(\epsilon(a_x+sh_x)\left((\bar{\bar{u}}+\tilde{u})_x - \frac{a_x+sh_x}{h}\tilde{u}_s\right)\right)_s.$$

We now integrate the last equation with respect to s from 0 to 1. Noting that $\int_0^1 \tilde{u}\,ds = 0$, $\int_0^1 h\,ds = h$, $\int_0^1 \bar{\bar{u}}\,ds = \bar{\bar{u}}$, and that ϕ is zero at both $s=0$ and $s=1$, we find that the left-hand side of the integrated expression is

$$((h\bar{\bar{u}})_t + (h\bar{\bar{u}}^2)_x + \left(\int_0^1 h\tilde{u}^2\,ds\right)_x + C^2h(a_x+h_x) = ...$$

Integrating the right hand side we obtain

$$\begin{aligned}
... &= \frac{\epsilon}{\lambda^2}\left[\frac{\tilde{u}_s(1)}{h} - \frac{\tilde{u}_s(0)}{h}\right] + (h\epsilon\bar{\bar{u}}_x)_x - \left[\int_0^1 \epsilon(a_x+sh_x)\tilde{u}_s\,ds\right]_x - \\
&\qquad \left(\epsilon(a_x+sh_x)\left((\bar{\bar{u}}+\tilde{u})_x - \frac{a_x+sh_x}{h}\tilde{u}_s\right)\right)\Big|_0^1 \\
... &= \frac{\epsilon}{\lambda^2}\left[\frac{\tilde{u}_s(1)}{h} - \frac{\tilde{u}_s(0)}{h}\right] + (h\epsilon\bar{\bar{u}}_x)_x - \left[\epsilon(a_x+h_x)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\right]_x \\
&\qquad - \epsilon\left[(a_x+h_x)\left((\bar{\bar{u}}+\tilde{u})_x(1) - \frac{a_x+h_x}{h}\tilde{u}_s(1)\right) - a_x\left((\bar{\bar{u}}+\tilde{u})_x(0) - \frac{a_x}{h}\tilde{u}_s(0)\right)\right] \\
... &= \frac{\epsilon}{\lambda^2 h}u_s(1) - \epsilon(a_x+h_x)\left(u_x(1) - \frac{a_x+h_x}{h}u_s(1)\right) - \frac{\epsilon}{\lambda^2 h}u_s(0) + \epsilon a_x\left(u_x(0) - \frac{a_x}{h}u_s(0)\right) \\
&\qquad + (h\epsilon\bar{\bar{u}}_x)_x - \left[\epsilon(a_x+h_x)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\right]_x \\
... &= \frac{\epsilon}{\lambda^2}K_{\mathrm{w}}(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{u}(1)) - \frac{\epsilon}{\lambda^2}hK_{\mathrm{fr}}(\bar{\bar{u}} + \tilde{u}(0)) + (h\epsilon\bar{\bar{u}}_x)_x - \left[\epsilon(a_x+h_x)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\right]_x.
\end{aligned}$$

Thus, the depth-averaged x-momentum equation is

$$(h\bar{\bar{u}})_t + (h\bar{\bar{u}}^2)_x + \left(\int_0^1 h\tilde{u}^2\,ds\right)_x + C^2 h(a_x + h_x) = \frac{\epsilon}{\lambda^2}K_{\mathrm{w}}(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{u}(1)) - \frac{\epsilon}{\lambda^2}hK_{\mathrm{fr}}\left(\bar{\bar{u}} + \tilde{u}(0)\right) + \left(h\epsilon\bar{\bar{u}}_x\right)_x - \Big[\epsilon\left(a_x + h_x\right)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\Big]_x . \tag{41}$$

6.3 Residual x-Momentum Equation

To obtain the equation for $\tilde{u}$, we subtract the depth-averaged x-momentum equation (41) from (34). The result is

$$\begin{aligned}(h\tilde{u})_t =& -2\left(h\tilde{u}\bar{\bar{u}}\right)_x - \left(h\tilde{u}^2\right)_x + \left(\int_0^1 h\tilde{u}^2\,ds\right)_x - \phi_s\bar{\bar{u}} - (\phi\tilde{u})_s - \frac{\epsilon}{\lambda^2}\left[K_{\mathrm{w}}^x\left(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{u}(1)\right)\right] \\ & \frac{\epsilon}{\lambda^2}\left[-hK_{\mathrm{fr}}^x\left(\bar{\bar{u}} + \tilde{u}(0)\right)\right] + \left[\epsilon\left(a_x + h_x\right)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\right] + \frac{\epsilon}{\lambda^2 h}\left(\tilde{u}_s\right)_s + \left(h\epsilon\tilde{u}_x\right)_x - \\ & \left(\epsilon\left(a_x + sh_x\right)\tilde{u}_s\right)_x - \left(\epsilon\left(a_x + sh_x\right)\left(\left(\bar{\bar{u}} + \tilde{u}\right)_x - \frac{a_x + sh_x}{h}\tilde{u}_s\right)\right)_s .\end{aligned} \tag{42}$$

We let

$$D = \frac{\epsilon}{\lambda^2},$$

define the function G by

$$\begin{aligned}G =& -h_t\tilde{u} - 2\left(h\tilde{u}\bar{\bar{u}}\right)_x - \left[h\left(\tilde{u}^2\right) - h\int_0^1 \tilde{u}^2(x,\eta,t)\,d\eta\right]_x - \phi_s\bar{\bar{u}} - (\phi\tilde{u})_s \\ & + \left[\epsilon\left(a_x + h_x\right)\tilde{u}(1) - \epsilon a_x\tilde{u}(0)\right] + \left(h\epsilon\tilde{u}_x\right)_x - \left(\epsilon\left(a_x + sh_x\right)\tilde{u}_s\right)_x .\end{aligned}$$

and note that we have selected the terms for G from (42) so that

$$\int_0^1 G(x,s,t)\,ds = 0.$$

The equation for $\tilde{u}$ then becomes

$$\begin{aligned}h\tilde{u}_t =& G + \frac{D}{h}\left[\left(1 + \lambda^2\left(a_x + sh_x\right)^2\right)\tilde{u}_s\right]_s - \left[\epsilon\left(a_x + sh_x\right)\left(\bar{\bar{u}}_x + \tilde{u}_x\right)\right]_s \\ & - D\left[K_{\mathrm{w}}\left(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{u}(1)\right) - hK_{\mathrm{fr}}\left(\bar{\bar{u}} + \tilde{u}(0)\right)\right].\end{aligned} \tag{43}$$

6.4 Equations for Boundary Conditions

On the lower surface $s = 0$, (35) becomes

$$\frac{D}{h}\left(1 + \lambda^2 a_x^2\right)\tilde{u}_s(0) - \left(\epsilon a_x\left(\bar{\bar{u}} + \tilde{u}(0)\right)\right) = DhK_{\mathrm{fr}}^x\left(\bar{\bar{u}} + \tilde{u}(0)\right) \tag{44}$$

and on the upper surface at $s = 1$, (36) becomes

$$\frac{D}{h}\left(1+\lambda^2\left(a_x+h_x\right)^2\right)\tilde{u}_s(1) - \left(\epsilon\left(a_x+h_x\right)\left(\bar{\tilde{u}}+\tilde{u}(1)\right)\right) = DK_{\mathrm{w}}^x\left(u_{\mathrm{w}}-\bar{\tilde{u}}-\tilde{u}(1)\right) \tag{45}$$

In the following section, we make the assumption that

$$D = \frac{\epsilon}{\lambda^2} \gg 1.$$

6.5 Approximating the x-Velocity Residual

The derivation of the approximate equation for $\tilde{u}$ follows the approach of Greenberg in [3]. The approximate equation is obtained by solving the velocity residual (43) with the boundary conditions (44) and (45) and exploiting the fact that D is much larger than both ϵ and 1 and that $D\lambda^2$ is much smaller than D. We retain only terms on the right hand side of (43), (44), and (45) that are O(D) while ignoring terms that are O(1), O(ϵ), and O($D\lambda^2$). Thus, we investigate the following system of equations

$$h\tilde{u}_t = \frac{D}{h}\tilde{u}_{ss} - D\left[K_{\mathrm{w}}\left(u_{\mathrm{w}}-\bar{\tilde{u}}-\tilde{u}(1)\right) - hK_{\mathrm{fr}}\left(\bar{\tilde{u}}+\tilde{u}(0)\right)\right], \tag{46}$$

$$\tilde{u}_s(0) = h^2K_{\mathrm{fr}}\left(\bar{\tilde{u}}+\tilde{u}(0)\right), \tag{47}$$

$$\tilde{u}_s(1) = hK_{\mathrm{w}}\left(u_{\mathrm{w}}-\bar{\tilde{u}}-\tilde{u}(1)\right). \tag{48}$$

The solution to (46)-(48) may be written as

$$\tilde{u} = \tilde{U} + w$$

where $\tilde{U}$ is an approximate steady state solution satisfying

$$\tilde{U}_{ss} = \left(K_{\mathrm{w}}h(u_{\mathrm{w}}-\bar{\tilde{u}}) - K_{\mathrm{fr}}h^2\bar{\tilde{u}} - K_{\mathrm{w}}h\tilde{U}(1) - K_{\mathrm{fr}}h^2\tilde{U}(0)\right), \tag{49}$$

$$\tilde{U}_s(0) = K_{\mathrm{fr}}h^2\left(\bar{\tilde{u}}+\tilde{U}(0)\right), \tag{50}$$

$$\tilde{U}_s(1) = K_{\mathrm{w}}h\left(u_{\mathrm{w}}-\bar{\tilde{u}}-\tilde{U}(1)\right) \tag{51}$$

and w is the transient component which may be written as

$$w(x,t) = \sum_{i=1}^{\infty} w_k e^{(-\mu_k t)}\phi^k(s)$$

where ϕ^k and μ_k satisfy the eigenvalue problem

$$\frac{D}{h^2}\left(\left(\phi_s^k\right)_s(s) + K_{\mathrm{w}}h\phi^k(1) + K_{\mathrm{fr}}h^2\phi^k(0)\right) + \mu_k\phi^k(s) = 0, \qquad 0 < s < 1$$
$$\phi_s^k(0) = K_{\mathrm{fr}}h^2\phi^k(0),$$
$$\phi_s^k(1) = -K_{\mathrm{w}}h\phi^k(1).$$

These eigen-functions satisfy

$$\int_0^1 \phi^k(s)\,ds = 0, \qquad k = 1, 2, 3, ...$$
$$\int_0^1 \phi^k(s)\phi^j(s)\,ds = 0. \qquad k = 1, 2, 3, ...$$

If we normalize $\phi^k(s)$ to satisfy

$$\int_0^1 \left(\phi^k\right)^2(s)\,ds = 1$$

we find that

$$0 < \mu_k = \frac{D}{h^2}\int_0^1 \left(\phi_s^k\right)^2(s)\,ds + \frac{D}{h^2}\left(K_{\rm w}h\left(\phi^k(1)\right)^2 + K_{\rm fr}h^2\left(\phi^k(0)\right)^2\right).$$

Since the only solution to

$$\begin{aligned}
&(\phi_s)_s + \left(K_{\rm w}h\phi(1) + K_{\rm fr}h^2\phi(0)\right) = 0,\\
&\phi_s(0) = K_{\rm fr}h^2\phi(0),\\
&\phi_s(1) = -K_{\rm w}h\phi(1)
\end{aligned}$$

is $\phi(s) \equiv 0$, we are guaranteed that $\mu = 0$ is not an eigen-value. Our assumption that $D = \epsilon/\lambda^2 \gg 1$ guarantees that solutions to the transient problem decay rapidly for any initial condition. This allows us to use the steady state solution $\tilde{U}$ as an approximation for $\tilde{u}$. Therefore, terms in the depth-averaged x-momentum equation that contain $\tilde{u}$ can be replaced with $\tilde{U}$.

The solution to the steady-state component of the residual (49)-(51) may be written as

$$\tilde{U} = \tilde{U}(0) + K_{\rm fr}\left(\bar{\tilde{u}} + \tilde{U}(0)\right)s + \left(K_{\rm w}h\left(u_{\rm w} - \bar{\tilde{u}}\right) - K_{\rm fr}h^2\bar{\tilde{u}} - K_{\rm w}h\tilde{U}(1) - K_{\rm fr}h^2\tilde{U}(0)\right)\frac{s^2}{2} \quad (52)$$

where

$$\tilde{U}(0) = -\frac{-K_{\rm w}h(u_{\rm w} - \bar{\tilde{u}})}{6\mathcal{B}} - \frac{\left(1 + K_{\rm w}h/4\right)K_{\rm fr}h^2\bar{\tilde{u}}}{6\mathcal{B}}, \quad (53)$$

$$\tilde{U}(1) = \frac{\left(1/3 + K_{\rm fr}h^2/12\right)K_{\rm w}h\left(u_{\rm w} - \bar{\tilde{u}}\right)}{\mathcal{B}} + \frac{K_{\rm fr}h^2\bar{\tilde{u}}}{6\mathcal{B}}, \quad (54)$$

and

$$\mathcal{B} = 1 + \frac{\left(K_{\rm w}h + K_{\rm fr}h^2\right)}{3} + \frac{K_{\rm w}K_{\rm fr}h^3}{12} > 0. \quad (55)$$

6.6 Summary of Depth-Averaged Equations

There are two depth-averaged equations of motion in our problem, summarized in the box below. Recall that $\tilde{U}$ is the steady state approximation to $\tilde{u}$ which we will use in subsequent computations. $\tilde{U}$ does not depend on x or t directly, instead it depends on $\bar{\bar{u}}$, u_{w}, and h, and those values depend on x and t. In the next section we solve the two partial differential equations of motion using numerical methods.

Mass Continuity

$$h_t + \left(h\bar{\bar{u}}\right)_x = 0 \tag{39}$$

x-Momentum

$$\begin{aligned}(h\bar{\bar{u}})_t + (h\bar{\bar{u}}^2)_x + \left(\int_0^1 h\tilde{U}^2\,ds\right)_x + C^2 h(a_x + h_x) = \\ \tfrac{\epsilon}{\lambda^2}K_{\mathrm{w}}(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{U}(1)) - \tfrac{\epsilon}{\lambda^2}hK_{\mathrm{fr}}\left(\bar{\bar{u}} + \tilde{U}(0)\right) + \left(h\epsilon\bar{\bar{u}}_x\right)_x \\ - \left[\epsilon\left(a_x + h_x\right)\tilde{U}(1) - \epsilon a_x\tilde{U}(0)\right]_x\end{aligned} \tag{41}$$

Steady State Approximation of Velocity Residual

$$\begin{aligned}\tilde{U} = \\ \tilde{U}(0) + K_{\mathrm{fr}}\left(\bar{\bar{u}} + \tilde{U}(0)\right)s + \left(K_{\mathrm{w}}h\left(u_{\mathrm{w}} - \bar{\bar{u}}\right) - K_{\mathrm{fr}}h^2\bar{\bar{u}} - K_{\mathrm{w}}h\tilde{U}(1) - K_{\mathrm{fr}}h^2\tilde{U}(0)\right)\tfrac{s^2}{2}\end{aligned} \tag{52}$$

$$\tilde{U}(0) = -\frac{-K_{\mathrm{w}}h(u_{\mathrm{w}} - \bar{\bar{u}})}{6\mathcal{B}} - \frac{(1 + K_{\mathrm{w}}h/4)K_{\mathrm{fr}}h^2\bar{\bar{u}}}{6\mathcal{B}} \tag{53}$$

$$\tilde{U}(1) = \frac{\left(1/3 + K_{\mathrm{fr}}h^2/12\right)K_{\mathrm{w}}h(u_{\mathrm{w}} - \bar{\bar{u}})}{\mathcal{B}} + \frac{K_{\mathrm{fr}}h^2\bar{\bar{u}}}{6\mathcal{B}} \tag{54}$$

$$\mathcal{B} = 1 + \frac{\left(K_{\mathrm{w}}h + K_{\mathrm{fr}}h^2\right)}{3} + \frac{K_{\mathrm{w}}K_{\mathrm{fr}}h^3}{12} \tag{55}$$

7 Finite-Dimensional Approximating Equations

Until now, we have developed a continuum description of the fluid flow where all fields, particularly $\bar{\bar{u}}$ and h, are defined at all points x and at all times t where the water column $h(x,t)$ is positive. This continuum problem is infinite dimensional—for example, a Fourier series representation of the solution will, in general, need infinitely many Fourier coefficients. Because the system formed by (39), (41), and (52)-(55) is non-linear, obtaining an exact solution is impossible. Therefore, this section presents a method for obtaining an approximate solution based on a finite-dimensional approximation of the infinite-dimensional problem.

We use a Lagrangian approach to create this approximate solution. With this method we define a series of "fluid-marker points" that we track for all time. In the context of this model, the term fluid-marker point does not refer to any specific particle at any place in the fluid. Instead it refers to the projection of a column of fluid onto the model's "grid". The grid is a structure for describing the lateral space of the fluid domain. In the two-dimensional model, the lateral space of the domain has one degree of freedom and the fluid-marker points are projections of a vertical column of water onto this line of lateral space.

We assume the wetted fluid region of the continuum problem has a lower bound of a and an upper bound of b at $t = 0$ so that $h(x,0)$ satisfies the following constraints

$$h(x,0) = \begin{cases} 0, & x < a, \\ h^o(x) > 0, & a < x < b, \\ 0, & b < x. \end{cases}$$

We create a grid that divides the region between a and b into $N+1$ points. If this grid is uniformly spaced then the i$^{\text{th}}$ point is denoted by

$$x_i^o = a + \frac{i-1}{N}(b-a), \qquad 1 \le i \le N+1.$$

Not all grids will be uniformly spaced; depending on the initial conditions, we may find it preferable to distribute grid points by other means. However, no matter the method of dividing the lateral space, there will always be $N+1$ grid points that are identified as fluid-marker points. The endpoints of this grid match the edges of the continuum problem where $x_1^o = a$ and $x_{N+1}^o = b$. The difference between two grid points at $t = 0$ is

$$x_{i+1}^o - x_i^o = \frac{b-a}{N}, \qquad 1 \le i \le N.$$

Given the sequence of grid points $\{x_i^o\}_{i=1}^{N+1}$ and the function for the initial water column

$h(x,0)$, we define the sequence $\{M_i^o\}_{i=1}^{N+1}$ by

$$M_i^o = \begin{cases} 0, & i = 1, \\ \int_{x_1^o}^{x_i^o} h^o(x)\,dx, & 2 \le i \le N+1. \end{cases}$$

The numbers M_i^o represent the amount of fluid in the interval $[x_1^0, x_i^0]$ at time $t = 0$ which we refer to as "fluid elements". We compute the amount of fluid in each interval $[x_i^0, x_{i+1}^0)$ by

$$m_i^o = M_{i+1}^o - M_i^o, \qquad 1 \le i \le N$$

and the average height of the water in the same interval by

$$h_i^o = \frac{m_i^o}{x_{i+1}^o - x_i^o} = \frac{N}{b-a} m_i^o. \tag{56}$$

Now that the height of the water column in each interval is approximated by the average water column, we no longer have any singularities in our governing equations along the free boundary where $h = 0$.

Given any domain of fluid described by a continuum mechanics model at a time $t = 0$, we have a finite-dimensional or discrete approximation of that domain. Our discretization will be a set of ordinary differential equations that tracks through time the position of the grid points $\{x_i^o\}_{i=1}^{N+1}$, the velocities of these fluid-marker points, and the average height of the water column between successive marker points. We adopt the following notation to denote these three values:

(1) $x_i(t)$ is the position at time t of the marker point that was located at x_i^o at $t = 0$ for $1 \le i \le N+1$.

(2) $\bar{\bar{u}}_i(t)$ is the velocity of the marker point $x_i(t)$ at time t for $1 \le i \le N+1$ and represents the value of $\bar{\bar{u}}(x_i(t), t)$.

(3) $h_i(t)$ is the average height of the water column in the interval $[x_i(t), x_{i+1}(t))$ for $1 \le i \le N$.

Note that x and $\bar{\bar{u}}$ satisfy

$$\frac{dx_i}{dt} = \bar{\bar{u}}_i(t), \qquad 1 \le i \le N,$$

and

$$x_i(0) = x_i^0, \qquad 1 \le i \le N.$$

7.1 Ordinary Differential Equation Approximation

Our strategy is to replace the partial differential equations in (39) and (41) by an approximating system of $2N + 2$ ordinary differential equations. At each of the points x_i, from $i = 1$ to $i = N + 1$, we create a pair of ordinary differential equations of the form

$$\frac{dx_i}{dt} = \bar{\bar{u}}_i,$$

and

$$\frac{1}{8}\left[m^o_{i-1}\frac{d\bar{\bar{u}}_{i-1}}{dt} + 3\left(m^o_{i-1} + m^o_i\right)\frac{d\bar{\bar{u}}_i}{dt} + m^o_i\frac{d\bar{\bar{u}}_{i+1}}{dt}\right] = F(x_i, \bar{\bar{u}}_i),$$

where m_i is a constant mass coefficient and $F(x_i, \bar{\bar{u}}_i)$ represents forcing terms. This creates $N + 1$ pairs of ordinary differential equations, each corresponding to the different index values of i. All $2N + 2$ equations are expressed more succinctly in vector form. We define two vectors

$$\mathbf{x} = \{x_j\}_{j=1}^{N+1}, \tag{57}$$

and

$$\mathbf{u} = \{\bar{\bar{u}}_j\}_{j=1}^{N+1} \tag{58}$$

and rewrite our approximating ordinary differential equations as

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}, \tag{59}$$

$$\hat{\mathbf{M}}\frac{d\mathbf{u}}{dt} = \mathbf{F}(\mathbf{x}, \mathbf{u}), \tag{60}$$

where $\hat{\mathbf{M}}$ is a tri-diagonal matrix and $\mathbf{F}$ is a vector of forcing terms. We calculate $\hat{\mathbf{M}}$ and $\mathbf{F}$ in the following sections.

7.2 Discretizing the Depth-Averaged Mass Continuity Equation

We integrate the continuum form of the averaged mass continuity equation (39) over the interval $[x_i, x_{i+1}]$. Assuming that all instances of h_i are evaluated at t, the result is

$$\begin{aligned} 0 &= \int_{x_i(t)}^{x_{i+1}(t)} h_t\, dx + \int_{x_i(t)}^{x_{i+1}(t)} \left(h\bar{\bar{u}}\right)_x\, dx \\ &= \frac{d}{dt}\int_{x_i(t)}^{x_{i+1}(t)} h\, dx - \left(x_{i+1}\right)_t h_i(x_{i+1}) + \left(x_i\right)_t h_i(x_i) + \bar{\bar{u}}(x_{i+1}, t)h_i(x_{i+1}) - \bar{\bar{u}}(x_i, t)h_i(x_i). \end{aligned}$$

Since $\left(x_i(t)\right)_t$ is equal to $\bar{\bar{u}}(x_i(t), t)$ the above expression reduces to

$$\frac{d}{dt}\int_{x_i(t)}^{x_{i+1}(t)} h\, dx = 0. \tag{61}$$

The integral of the water column over an interval between fluid markers is the amount of fluid in that interval, or, since we assume that density is a constant $\rho = 1$, the mass of that interval . From (61) we see that this mass does not change over time so we generalize that $m_i^o = m_i$. We see that

$$\int_{x_i(t)}^{x_{i+1}(t)} h(x,t)\,dx = \int_{x_i^o}^{x_{i+1}^o} h(x,0)\,dx = \int_{x_i^o}^{x_{i+1}^o} h^o(x)\,dx = m_i^o$$

and (56) implies that

$$m_i = (x_{i+1}(t) - x_i(t))\,h_i(t). \tag{62}$$

This is the discrete approximation to the depth-averaged continuity equation.

7.3 Discretizing the Depth-Averaged x-Momentum Equation

We have established a grid scheme that discretizes the fluid domain. Now we derive a set of evolution equations for the sequence of velocities $\{\bar{\bar{u}}_i(t)\}_{i=1}^{N+1}$ which are consistent with the depth-averaged x-momentum equation. To obtain these equations we extend the discrete velocities $\{\bar{\bar{u}}_i(t)\}_{i=1}^{N+1}$ and average water columns $\{h_i\}_{i=1}^{N}$ to fields defined on the interval $(-\infty, \infty)$. For $\bar{\bar{u}}_i$, we do this by piecewise linear extension:

$$\bar{\bar{u}}(x,t) = \begin{cases} \bar{\bar{u}}_i(t), & x \le x_1(t), \\ \bar{\bar{u}}_i(t) + \left(\frac{\bar{\bar{u}}_{i+1}(t) - \bar{\bar{u}}_i(t)}{x_{i+1}(t) - x_i(t)}\right)(x - x_i(t)), & x_i(t) \le x \le x_{i+1}(t) \text{ and } 1 \le i \le N, \\ \bar{\bar{u}}_{N+1}(t), & x_{N+1}(t) \le x. \end{cases} \tag{63}$$

For the average water columns, we do this by a piecewise constant function

$$h(x,t) = \begin{cases} 0, & x < x_1(t), \\ h_i(t), & x_i(t) < x < x_{i+1}(t) \text{ and } 1 \le i \le N, \\ 0, & x_{N+1}(t) < x. \end{cases} \tag{64}$$

where $h_i(t) = \frac{m_i^o}{x_{i+1}(t) - x_i(t)}$.

To obtain the evolution equations from the discrete velocities $\{\bar{\bar{u}}_i(t)\}_{i=1}^{N+1}$, we integrate the depth-averaged x-momentum equation around each point x_i. We choose to integrate from the midpoints between adjacent fluid markers. We let

$$x_{i-1/2} = \tfrac{x_i + x_{i-1}}{2} \quad \text{and} \quad x_{i+1/2} = \tfrac{x_{i+1} + x_i}{2},$$

be the lower and upper bounds of this region of integration and a corresponding notation for $\bar{\bar{u}}$ is

$$\bar{\bar{u}}_{i+1/2} = \bar{\bar{u}}\big|_{x_{i+1/2}},$$
$$\bar{\bar{u}}_{i-1/2} = \bar{\bar{u}}\big|_{x_{i-1/2}}.$$

The depth-averaged x-momentum equation (41) can be reorganized as

$$\overbrace{(h\bar{\bar{u}})_t + (h\bar{\bar{u}}^2)_x}^{\text{momentum}} = \overbrace{-\left(\int_0^1 h\tilde{U}^2\,ds\right)_x - \frac{C^2}{2}(h^2)_x}^{\text{pressure}} - \overbrace{C^2 h a_x}^{\text{potential}}$$
$$+ \underbrace{\frac{\epsilon}{\lambda^2}K_{\text{w}}(\bar{\bar{u}}_{\text{w}} - \bar{\bar{u}} - \tilde{U}(1)) - \frac{\epsilon}{\lambda^2}hK_{\text{fr}}\left(\bar{\bar{u}} + \tilde{U}(0)\right)}_{\text{wind and friction}} + \underbrace{(h\epsilon\bar{\bar{u}}_x)_x}_{\text{viscosity}} - \underbrace{\left[\epsilon\left(a_x + h_x\right)\tilde{U}(1) - \epsilon a_x\tilde{U}(0)\right]_x}_{\text{remainder}}.$$

For simplicity, we choose to ignore the remainder term because the remainder term is much smaller than the other terms in the depth-averaged x-momentum equation. We integrate around each point x_i to obtain

$$\overbrace{\int_{x_{i-1/2}}^{x_{i+1/2}}\left((h\bar{\bar{u}})_t + (h\bar{\bar{u}}^2)_x\right)\,dx}^{\text{momentum}} = \overbrace{-C^2\int_{x_{i-1/2}}^{x_{i+1/2}} h a_x\,dx}^{\text{potential}} - \overbrace{\int_{x_{i-1/2}}^{x_{i+1/2}}\left[C^2\left(\frac{h^2}{2}\right) + h\int_0^1\tilde{U}^2\,ds\right]_x dx}^{\text{pressure}}$$
$$+ \underbrace{\int_{x_{i-1/2}}^{x_{i+1/2}}(\epsilon h\bar{\bar{u}}_x)_x\,dx}_{\text{viscosity}} + \underbrace{\int_{x_{i-1/2}}^{x_{i+1/2}}\frac{\epsilon}{\lambda^2}\left[K_{\text{w}}(u_{\text{w}} - \bar{\bar{u}} - \tilde{U}(1)) - hK_{\text{fr}}(\bar{\bar{u}} + \tilde{U}(0))\right]dx}_{\text{wind and friction}} \tag{65}$$

7.3.1 Momentum Terms

The momentum terms from the integrated averaged x-momentum equation (65) can be written as

$$\frac{d}{dt}\int_{x_{i-1/2}}^{x_{i+1/2}}(h\bar{\bar{u}})\,dx - h_i\bar{\bar{u}}_{i+1/2}\left(x_{i+1/2}\right)_t + h_{i-1}\bar{\bar{u}}_{i-1/2}\left(x_{i-1/2}\right)_t + h_i\left(\bar{\bar{u}}_{i+1/2}\right)^2 - h_{i-1}\left(\bar{\bar{u}}_{i-1/2}\right)^2.$$

Recalling that $\frac{dx}{dt} = \bar{\bar{u}}$, this last expression reduces to

$$\frac{d}{dt}\int_{x_{i-1/2}}^{x_{i+1/2}}(h\bar{\bar{u}})\,dx. \tag{66}$$

Using (62)-(64) we see that

$$\begin{aligned}\int_{x_{i-1/2}}^{x_{i+1/2}}(h\bar{\bar{u}})\,dx &= h_i\int_{x_i}^{x_{i+1/2}}\bar{\bar{u}}\,dx + h_{i-1}\int_{x_{i-1/2}}^{x_i}\bar{\bar{u}}\,dx \\ &= h_i\left[\left(\frac{\frac{\bar{\bar{u}}_{i+1}+\bar{\bar{u}}_i}{2} + \bar{\bar{u}}_i}{2}\right)\frac{x_{i+1}-x_i}{2}\right] + h_{i-1}\left[\left(\frac{\bar{\bar{u}}_i + \frac{\bar{\bar{u}}_i+\bar{\bar{u}}_{i-1}}{2}}{2}\right)\frac{x_i - x_{i-1}}{2}\right] \\ &= \frac{1}{8}\left[\bar{\bar{u}}_{i-1}m_{i-1} + 3\bar{\bar{u}}_i\left(m_{i-1}+m_i\right) + \bar{\bar{u}}_{i+1}m_i\right].\end{aligned}$$

Therefore, (66) can be rewritten in vector form as

$$\mathbf{M}\left(\frac{d\mathbf{u}}{dt}\right) \tag{67}$$

where $\mathbf{M}$ is a tri-diagonal mass matrix with elements defined by

$$\begin{aligned}
&\mathbf{M}_{1,1} = \frac{3}{8}m_1, \\
&\mathbf{M}_{N+1,N+1} = \frac{3}{8}m_N, \\
&\mathbf{M}_{j,j} = \frac{3}{8}(m_{j-1} + m_j), && 2 \le j \le N, \\
&\mathbf{M}_{j,j-1} = \frac{1}{8}m_{j-1}, && 2 \le j \le N+1, \\
&\mathbf{M}_{j,j+1} = \frac{1}{8}m_j, && 1 \le j \le N.
\end{aligned}$$

7.3.2 Potential and Pressure Terms

The potential term from the integrated, depth-averaged x-momentum equation (65) can be written as

$$C^2 h_{i-1}\left(a(x_i) - a(x_{i-1/2})\right) + C^2 h_i\left(a(x_{i+1/2}) - a(x_i)\right). \tag{68}$$

The pressure terms are

$$\frac{C^2}{2}h_i^2 - \frac{C^2}{2}h_{i-1}^2 + h_i\int_0^1 \tilde{U}(x_{i+1/2})^2\,ds - h_{i-1}\int_0^1 \tilde{U}(x_{i-1/2})^2\,ds. \tag{69}$$

These terms can be evaluated in the forms above. We represent their vector forms by $\mathbf{F}_{\text{pres}}$ and $\mathbf{F}_{\text{pot}}$.

7.3.3 Viscosity Term

In evaluating the viscosity term, we assume that ϵ is piecewise constant in the intervals between fluid markers, so that

$$\epsilon(x,t) \equiv \epsilon_i, \quad x_i < x < x_{i+1} \text{ and } 1 \le i \le N.$$

Using this ϵ, we write the integrated viscosity term as

$$\epsilon_i h_i \frac{\bar{\bar{u}}_{i+1} - \bar{\bar{u}}_i}{x_{i+1} - x_i} - \epsilon_{i-1}h_{i-1}\frac{\bar{\bar{u}}_i - \bar{\bar{u}}_{i-1}}{x_i - x_{i-1}}$$

There are many possible choices for ϵ_i. One choice is

$$\epsilon_i = \frac{\mu}{8dt}\left(\frac{b-a}{N}\right)^2. \tag{70}$$

This value for ϵ has the advantage of being constant in both space and time. Note that dt is the size of the time step that we will be taking in our numerical computations. Another choice is

$$\epsilon_i = \mu \frac{(x_{i+1} - x_i)^2}{8dt}. \tag{71}$$

The viscous forces may be written in vector form as

$$\bar{\mathbf{M}}\mathbf{u} \tag{72}$$

where $\mathbf{u}$ is defined in (58) and $\bar{\mathbf{M}}$ is a tri-diagonal matrix defined by

$$\begin{aligned}
\bar{\mathbf{M}}_{1,1} &= -\frac{\epsilon_1 h_1}{\Delta x_1}, \\
\bar{\mathbf{M}}_{N+1,N+1} &= -\frac{\epsilon_N h_N}{\Delta x_N}, \\
\bar{\mathbf{M}}_{j,j} &= -\left(\frac{\epsilon_{j-1} h_{j-1}}{\Delta x_{j-1}} + \frac{\epsilon_j h_j}{\Delta x_j}\right), && 2 \le j \le N, \\
\bar{\mathbf{M}}_{j,j-1} &= \frac{\epsilon_{j-1} h_{j-1}}{\Delta x_{j-1}}, && 2 \le j \le N+1, \\
\bar{\mathbf{M}}_{j,j+1} &= \frac{\epsilon_j h_j}{\Delta x_j}, && 1 \le j \le N.
\end{aligned} \tag{73}$$

We are using the notation

$$\Delta x_i = (x_{i+1} - x_i).$$

7.3.4 Friction and Wind Terms

The wind term from the integrated averaged x-momentum equation (65) is

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\epsilon}{\lambda^2} \left[K_{\mathrm{w}} \left(u_{\mathrm{w}} - \bar{\bar{u}} - \tilde{U}(1) \right) \right] dx$$

and the friction term is

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\epsilon}{\lambda^2} \left[-h K_{\mathrm{fr}} \left(\bar{\bar{u}} + \tilde{U}(0) \right) \right] dx.$$

Using the definitions of $\tilde{U}(0)$ and $\tilde{U}(1)$ from (53)-(55), we expand these two terms to

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\epsilon}{\lambda^2} \left[K_{\mathrm{w}} \left(u_{\mathrm{w}} - \bar{\bar{u}} \right) - \frac{4 + K_{\mathrm{fr}} K_{\mathrm{w}}^2 h^2}{12\mathcal{B}} h \left(u_{\mathrm{w}} - \bar{\bar{u}} \right) - \frac{K_{\mathrm{fr}} K_{\mathrm{w}} h}{6\mathcal{B}} h \bar{\bar{u}} \right] dx$$

and

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\epsilon}{\lambda^2} \left[-K_{\mathrm{fr}} h \bar{\bar{u}} + \frac{K_{\mathrm{w}} K_{\mathrm{fr}} h}{6\mathcal{B}} h \left(u_{\mathrm{w}} - \bar{\bar{u}} \right) + \frac{\left(1 + K_{\mathrm{w}} h/4\right) K_{\mathrm{fr}}^2 h^2}{6\mathcal{B}} h \bar{\bar{u}} \right] dx.$$

Using (62), we evaluate these expressions to obtain the combined wind and friction term

$$
\begin{aligned}
(u_{\mathrm{w}}-\bar{\bar{u}})_{i-1}&\left[\frac{K_{\mathrm{w}}\epsilon_{i-1}}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i-1}+2\right)K_{\mathrm{fr}}h_{i-1}}{12\mathcal{B}}h_{i-1}\right)\Delta x_{i-1}\right]\\
&+(u_{\mathrm{w}}-\bar{\bar{u}})_{i}\,\frac{3K_{\mathrm{w}}}{8\lambda^2}\left[\epsilon_{i-1}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i-1}+2\right)K_{\mathrm{fr}}h_{i-1}}{12\mathcal{B}}h_{i-1}\right)\Delta x_{i-1}\right.\\
&\left.+\,\epsilon_i\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i}+2\right)K_{\mathrm{fr}}h_{i}}{12\mathcal{B}}h_{i}\right)\Delta x_{i}\right]\\
&+(u_{\mathrm{w}}-\bar{\bar{u}})_{i+1}\left[\frac{K_{\mathrm{w}}\epsilon_{i}}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i}+2\right)K_{\mathrm{fr}}h_{i}}{12\mathcal{B}}h_{i}\right)\Delta x_{i}\right]\\
&+\bar{\bar{u}}_{i-1}\left[\frac{K_{\mathrm{fr}}\epsilon_{i-1}}{8\lambda^2}\left(\frac{\left(1+K_{\mathrm{w}}h_{i-1}/4\right)K_{\mathrm{fr}}h_{i-1}-K_{\mathrm{w}}}{6\mathcal{B}}h_{i-1}-1\right)m_{i-1}\right]\\
&+\bar{\bar{u}}_{i}\frac{3K_{\mathrm{fr}}}{8\lambda^2}\left[\epsilon_{i-1}\left(\frac{\left(1+K_{\mathrm{w}}h_{i-1}/4\right)K_{\mathrm{fr}}h_{i-1}-K_{\mathrm{w}}}{6\mathcal{B}}h_{i-1}-1\right)m_{i-1}\right.\\
&\left.+\epsilon_i\left(\frac{\left(1+K_{\mathrm{w}}h_{i}/4\right)K_{\mathrm{fr}}h_{i}-K_{\mathrm{w}}}{6\mathcal{B}}h_{i}-1\right)m_{i}\right]\\
&+\bar{\bar{u}}_{i+1}\left[\frac{K_{\mathrm{fr}}\epsilon_{i}}{8\lambda^2}\left(\frac{\left(1+K_{\mathrm{w}}h_{i}/4\right)K_{\mathrm{fr}}h_{i}-K_{\mathrm{w}}}{6\mathcal{B}}h_{i}-1\right)m_{i}\right].
\end{aligned}
$$

To express this expression in vector form, we define two matrices $\mathbf{M}_{\mathrm{f}}$ and $\mathbf{M}_{\mathrm{w}}$. The matrices are

$$
\begin{aligned}
(\mathbf{M}_{\mathrm{w}})_{1,1}&=\frac{K_{\mathrm{w}}\epsilon_1}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_1+2\right)K_{\mathrm{fr}}h_1}{12\mathcal{B}}h_1\right)\Delta x_1,\\
(\mathbf{M}_{\mathrm{w}})_{N+1,N+1}&=\frac{K_{\mathrm{w}}\epsilon_N}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_N+2\right)K_{\mathrm{fr}}h_N}{12\mathcal{B}}h_N\right)\Delta x_N,\\
(\mathbf{M}_{\mathrm{w}})_{j,j}&=\frac{3K_{\mathrm{w}}}{8\lambda^2}\left[\epsilon_{i-1}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i-1}+2\right)K_{\mathrm{fr}}h_{i-1}}{12\mathcal{B}}h_{i-1}\right)\Delta x_{i-1}\right.\\
&\qquad\left.+\epsilon_i\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i}+2\right)K_{\mathrm{fr}}h_{i}}{12\mathcal{B}}h_{i}\right)\Delta x_{i}\right], \qquad 2\le j\le N,\\
(\mathbf{M}_{\mathrm{w}})_{j,j-1}&=\frac{K_{\mathrm{w}}\epsilon_{i-1}}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i-1}+2\right)K_{\mathrm{fr}}h_{i-1}}{12\mathcal{B}}h_{i-1}\right)\Delta x_{i-1}, \qquad 2\le j\le N+1,\\
(\mathbf{M}_{\mathrm{w}})_{j,j+1}&=\frac{K_{\mathrm{w}}\epsilon_{i}}{8\lambda^2}\left(1-\frac{4+\left(K_{\mathrm{w}}h_{i}+2\right)K_{\mathrm{fr}}h_{i}}{12\mathcal{B}}h_{i}\right)\Delta x_{i}, \qquad 1\le j\le N,
\end{aligned}
$$

and

$$
\begin{aligned}
(\mathbf{M}_{\text{fr}})_{1,1} &= \frac{K_{\text{fr}}\epsilon_1}{8\lambda^2}\left(\frac{(1+K_{\text{w}}h_1/4)\,K_{\text{fr}}h_1 - K_{\text{w}}}{6\mathcal{B}}h_1 - 1\right)m_1,\\
(\mathbf{M}_{\text{fr}})_{N+1,N+1} &= \frac{K_{\text{fr}}\epsilon_N}{8\lambda^2}\left(\frac{(1+K_{\text{w}}h_N/4)\,K_{\text{fr}}h_N - K_{\text{w}}}{6\mathcal{B}}h_N - 1\right)m_N,\\
(\mathbf{M}_{\text{fr}})_{j,j} &= \frac{3K_{\text{fr}}}{8\lambda^2}\left[\epsilon_{i-1}\left(\frac{(1+K_{\text{w}}h_{i-1}/4)\,K_{\text{fr}}h_{i-1} - K_{\text{w}}}{6\mathcal{B}}h_{i-1} - 1\right)m_{i-1}\right.\\
&\qquad\left.+\epsilon_i\left(\frac{(1+K_{\text{w}}h_i/4)\,K_{\text{fr}}h_i - K_{\text{w}}}{6\mathcal{B}}h_i - 1\right)m_i\right], \qquad 2\le j\le N,\\
(\mathbf{M}_{\text{fr}})_{j,j-1} &= \frac{K_{\text{fr}}\epsilon_{i-1}}{8\lambda^2}\left(\frac{(1+K_{\text{w}}h_{i-1}/4)\,K_{\text{fr}}h_{i-1} - K_{\text{w}}}{6\mathcal{B}}h_{i-1} - 1\right)m_{i-1}, \quad 2\le j\le N+1,\\
(\mathbf{M}_{\text{fr}})_{j,j+1} &= \frac{K_{\text{fr}}\epsilon_i}{8\lambda^2}\left(\frac{(1+K_{\text{w}}h_i/4)\,K_{\text{fr}}h_i - K_{\text{w}}}{6\mathcal{B}}h_i - 1\right)m_i, \qquad 1\le j\le N.
\end{aligned}
$$

We also define a vector

$$\mathbf{u}_{\text{w}} = \{(u_{\text{w}})_j\}_{j=1}^{N+1}.$$

With these definitions, the friction and wind terms may be written in vector form as

$$\mathbf{M}_{\text{w}}\left(\mathbf{u}_{\text{w}} - \mathbf{u}\right) - \mathbf{M}_{\text{fr}}\mathbf{u}. \tag{74}$$

7.3.5 Summary of All Terms of Depth-Averaged x-Momentum Equation

When the terms of (65) have been translated into vector form, the equation becomes

$$\overbrace{\mathbf{M}\left(\frac{d\mathbf{u}}{dt}\right)}^{\text{momentum}} = -\overbrace{\mathbf{F}_{\text{pres}}}^{\text{pressure}} - \overbrace{\mathbf{F}_{\text{pot}}}^{\text{potential}} + \overbrace{\tilde{\mathbf{M}}\mathbf{u}}^{\text{viscosity}} + \overbrace{\mathbf{M}_{\text{w}}\left(\mathbf{u}_{\text{w}} - \mathbf{u}\right) - \mathbf{M}_{\text{fr}}\mathbf{u}}^{\text{wind and friction}}. \tag{75}$$

This is the equation in the form of (60) that forms the second part of our ordinary differential equation approximation, the first part of which is

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}.$$

We solve this system of ordinary differential equations through operator splitting. Each time we step forward through time by one time step, we evaluate our system in two distinct parts. For the first part we hold $\mathbf{x}$ constant and update $\mathbf{u}$; in the second part we hold $\mathbf{u}$ constant and update $\mathbf{x}$. We use a time grid where the constant interval dt is the length of time between t^n and t^{n+1} for any n. Another way of stating this is that $t^n = n\,dt$ for $n = 0, 1, ...$ etc. We use a superscript notation to signify the time step at which a value is being calculated,

thus $\mathbf{u}^{n+1}$ refers to the depth-averaged velocity one time step after $\mathbf{u}^n$. The two steps can be described by

$$\begin{array}{ccc} \textbf{Step 1} & & \textbf{Step 2} \\ t^n \leq t \leq t^{n+1} & & t^n \leq t \leq t^{n+1} \\ \frac{d\mathbf{x}}{dt} = 0 & & \frac{d\mathbf{x}}{dt} = \mathbf{u} \\ \mathbf{M}\frac{d\mathbf{u}}{dt} = \mathcal{F}(\mathbf{x}, \mathbf{u}) & & \mathbf{M}\frac{d\mathbf{u}}{dt} = 0 \end{array} \tag{76}$$

Note that we do not consider $\mathbf{u}_{\mathrm{w}}$ a variable upon which (75) depends; it is given or externally supplied data like the mass, viscosity, and friction matrices.

8 Implementation of the Two-Dimensional Model

At the end of the previous section, we found a solution for the velocity fields of the two-dimensional model. This solution involves a "leap-frog" updating process and uses a Lagrangian method to track particles from time zero through all future time steps. Within each time step, we solve first for the depth averaged mean velocity, $\bar{\bar{u}}$, of the fluid at each particle location using data from the previous time step. Next we use the mean velocity to update the location of each particle, x. Finally, with a new array of particle locations and a constant array of masses, we can calculate the water column, h, in each interval between particles.

There are many options for implementing the solution of the final differential equation in MATLAB. Two of the easiest solutions are the explicit and implicit Euler methods. In explicit methods, such as the forward Euler method, the value of the variable of interest—in this case the depth averaged velocity, $\bar{\bar{u}}$—at the next time step is defined in terms of known values from the current time step. Implicit methods, such as the backwards Euler method, defines the variable at the next time step in terms of unknown values from the next time step. The variable of interest must then be solved for; this solution takes time but the advantage is that implicit solutions are usually more stable allowing the model to use larger time steps. The two-dimensional model has been implemented using both approaches.

The final differential equation (75) represents a series of n equations, one for each tracked particle. In the context of implementing this model, the advantage of the explicit Euler method is that with certain viscosity values each equation can be independent. That means that there is no need for linear algebra when solving; instead MATLAB will just solve a series of n independent equations. The implicit Euler method does require linear algebra because all the differential equations in the series are dependent on each other; but the advantage of this method is its increased stability. The implicit method produces stable solutions for almost all initial conditions when the ratio $\frac{dt}{dx}$ is 0.1 while the explicit method can fail if the spacing between the particle locations, dx, becomes too small. Testing of the two methods against each other have shown that the explicit and implicit methods provide near identical results when both are stable and initial conditions are the same. However, when the residual velocities and wind and surface friction forces are added, the domain of inputs on which the explicit solution is stable becomes much smaller unless the ratio $\frac{dt}{dx}$ is reduced, so the implicit method is preferred.

Whether the explicit or implicit method is used, MATLAB implementation of this leapfrog process takes as input the state of a region of water at any time and produces as output the state of that water at a future time. Here state is defined primarily by the three variables of interest to this program: depth averaged mean velocity at each particle, $\bar{\bar{u}}$, location of each particle, x, and water column of each fluid element, h. In order to perform this calculation, the program needs these three pieces of information at time zero.

In addition, the program needs a variety of other initial data, including a function defining bathymetry, a function defining wind, constants for wind friction and bottom friction, initial velocity of the fluid, and more.

The program provides two graphical outputs for visualizing results of the model. A plot of height over bathymetry versus x gives the user a view of what the two dimensional model might look like in a real physical setting–or at least as real as one can get with two dimensions. The other graph plots the center of mass versus the average momentum of the mass of water. This graphic is useful as a diagnostic. For cases where the bathymetry is a parabola and there is no friction, this graph will trace a circle as the mass of fluid settles into a stable path of oscillation.

The purpose of creating the two-dimensional model is to investigate the mathematics and physics of the shallow water equations before moving on to modeling the three-dimensional shallow water equations. Furthermore, because of the simplicity of the two-dimensional shallow water equations, this first model incorporates more features than will be implemented in the three-dimensional model within the scope of this project. Factors such as the residual velocities at each depth layer, wind friction, and variable viscosities have not been included in the three-dimensional model.

8.1 Operation of the Two-Dimensional Model

The two-dimensional model is implemented as a MATLAB function that can be run from the MATLAB base workspace without any input arguments. The initial conditions are specified through the use of another function call that is designed to be user-modifiable. The actual calculations are carried out within a loop in the main function of the model.

The most important single line of this function is derived directly from (75), which is the first step of the two-step operator splitting method established in (76). This first-step differential equation (75) is

$$\mathbf{M}\left(\frac{d\mathbf{u}}{dt}\right) = -\mathbf{F}_{\text{pres}} - \mathbf{F}_{\text{pot}} + \bar{\mathbf{M}}\mathbf{u} + \mathbf{M}_{\text{w}}\left(\mathbf{u}_{\text{w}} - \mathbf{u}\right) - \mathbf{M}_{\text{fr}}\left(\mathbf{u}\right).$$

Since the two-dimensional model solves this equation using an implicit scheme, the derivative of $\mathbf{u}$ is approximated by the backwards Euler method. Using this approximation, (75) becomes

$$\mathbf{M}\left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^{n}}{dt}\right) = -\mathbf{F}_{\text{pres}} - \mathbf{F}_{\text{pot}} + \left(\bar{\mathbf{M}} - \mathbf{M}_{\text{w}} - \mathbf{M}_{\text{fr}}\right)\mathbf{u}^{n+1} + \mathbf{M}_{\text{w}}\mathbf{u}_{\text{w}}$$

where superscripts represent the time step at which a variable is evaluated. Solving this equation for $\mathbf{u}^{n+1}$ yields

$$\left[\mathbf{M} - dt\left(\bar{\mathbf{M}} - \mathbf{M}_{\text{w}} - \mathbf{M}_{\text{fr}}\right)\right]\mathbf{u}^{n+1} = dt\left[-\mathbf{F}_{\text{pres}} - \mathbf{F}_{\text{pot}} + \mathbf{M}_{\text{w}}\mathbf{u}_{\text{w}} + \mathbf{M}\mathbf{u}^{n}\right].$$

This equation is used to compute the velocity of each tracked particle in a certain time step using the velocity of the previous time step. The various matrices for mass, viscosity, and friction coefficients must be created within each time step previous to evaluating this equation.

When implemented in MATLAB, the value of $\mathbf{u}^{n+1}$ is determined using linear algebra; specifically, MATLAB's backsolve feature. The code that solves this equation is

```
U = ((MM - MMVV + dt*(MMWW + MMFF)) \ (-dt*POT' - dt*PRES' + ...
    dt*MMWW*UW' + MM*U'))';.
```

In this code, `U` represents the vector of velocities for all tracked particles. It is updated with new values when this line of code finishes execution. The matrices $\mathbf{M}$, $\bar{\mathbf{M}}$, $\mathbf{M}_{\text{w}}$, and $\mathbf{M}_{\text{fr}}$ correspond to `MM`, `MMVV`, `MMWW`, and `MMFF`, respectively. The force vectors $\mathbf{F}_{\text{pres}}$ and $\mathbf{F}_{\text{pot}}$ become `PRES` and `POT`. The one discrepancy between the equation and the implementation is that the viscosity matrix–$\bar{\mathbf{M}}$ and `MMVV`–is multiplied by dt in the equation but not in the implementation. This is because $\bar{\mathbf{M}}$ as defined in (73) is already multiplied by a factor of dt. Rather than multiplying by this factor and then later dividing it out, it is more efficient to simply omit this factor in the code.

After solving for new values of `U` in each time step, we determine the vector of each tracked particle's position, represented in the code by `X`. This is the second step of (76). Recall that the differential equation we solve in this step is

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}.$$

This is implemented in the model as

```
X = X + U * dt;
```

After this step is complete, the last piece of information of interest to the model is the water column. When the method for changing from a system of continuum equations to discrete equations was presented in Section 7, the relationship between water column and mass within each fluid element was established in (56) to be

$$h_i = \frac{m_i^o}{x_{i+1} - x_i}.$$

This relationship is established at time zero at which time the mass, m_i^o, within each fluid element is fixed. For all other times this relationship still holds; h_i and $x_{i+1} - x_i$ change with each time step while m_i stays constant. Therefore, in the implementation of the model, the water column is calculated by the code

```
H = M./diff(X);.
```

The `diff` function in MATLAB computes the difference between adjacent elements of the vector `X`. For a vector holding the elements x_1 to x_{n+1}, `diff(X)` returns a length n vector where each element has the value $x_{i+1} - x_i$.

The three lines of code in this chapter are the heart of the implementation of the two-dimensional model. They lie within a "time-stepping loop" that counts up time steps from time zero when the program is started. The rest of the code within the loop is devoted to setting up the various matrices and forces needed to compute the particle velocities and then plotting the resulting data. Matrices $\mathbf{M}$ and $\bar{\mathbf{M}}$ are constant and are set at the beginning of the program before it enters the time-stepping loop. The pressure and potential terms are set in sections of the code labeled after them, while $\mathbf{M}_{\text{w}}$ and $\mathbf{M}_{\text{fr}}$ are set in the wind and friction section. Within the loop two indices are maintained: `t`, the time index which represents model time, and `i`, the counting index which is the number of iterations through which the loop has passed. Each time the counting index advances by 1, the time index advances by dt.

Visualizing the data has no real part to play in the actual operation of the two-dimensional model. The model will create numerical output representing the velocity and position of fluid particles and water column of fluid elements from any valid input and without the help of any plotting. However, visualization provides the human user with a way to immediately understand the data that is being generated from the model without having to read and compare huge vectors of raw numbers. As mentioned in the previous section, there are two graphical outputs from the two-dimensional model: the water column versus x plot and the center of mass versus average momentum plot. Plotting does not have to be carried out by the program during every single time step. Instead plotting is done at a specific time interval, which can be specified by the initialization file.

The initialization takes place before the program enters the time-stepping loop. A separate function is run that creates initial conditions based on user specifications. Through its development process, the initialization program has expanded to encompass more and more user-input so that now almost every aspect of the program is modifiable, from the magnitude of viscosity and friction, to the initial shape of the bathymetry and wetted area, to the manner in which functions are defined as inputs. The initialization file is run at the beginning of the model's executable and it loads its state date into a structure which is then passed to the model itself; the model updates this data structure as it steps through time. Once the model is done running, it compiles all the data back into the state structure which it returns as function output. This data structure can thus be used as input for another run of the model.

The complete code for the two-dimensional model is included in (C) and the initialization code is included in (B).

8.2 Conclusions from the Two-Dimensional Model

The two-dimensional model is a stepping stone that we created to gain better understanding of the mathematics needed for the three-dimensional model. Because the two-dimensional model has fewer governing equations it is simpler to trace through the series of transformations needed to bring this model to implementation. We experimented with our novel ideas on this simpler model before tackling the three-dimensional problem that is the ultimate goal of this Trident project.

For first few steps of transformation of the two-dimensional model's governing equations, we followed the precedent of the Princeton Ocean Model. The non-dimensionalization and s-Coordinate transformation both originated with POM. However, we successfully introduced several novel ideas in our derivation. One new concept is the way we handle the residuals after depth-averaging the conservation of mass and momentum equations in section 6. We replace the evolution equation by the local equilibrium solution, thereby removing a degree of freedom from our model. This will prove to be particularly useful for the more complicated three-dimensional model.

Another innovation is using a Lagrangian particle tracking approach for our implementation rather than the Eulerian method used in POM. By solving the problem on a domain with a boundary fixed by the locations of the fluid-marker points that we track, we obtain information about the free boundary. This makes our model ideal for determining if a location is wetted or not and also helps to make our model more accurate for shallow water estuaries and enclosed bays.

9 Moving from the Two Dimensions to Three Dimensions

We now move from the shallow water equations with one lateral dimension to those with two lateral dimensions. A model of these equations has already been created by J. M. Greenberg, an adviser on this project. The objective of this part of the project is to take this model—which is implemented in serial for one processor—and alter it so that it can run in parallel on multiple processors. To do this, the serial code must first be modified to make it amenable to efficient parallelization. This entails both dividing up the processor time required for the calculations in the model so that each processor has a roughly equal load and providing a means for communicating data between processors.

The mathematics of this model are similar to those of the two-dimensional equations from sections 3 through 7, except that there are now two lateral coordinates, x and y, in addition to the vertical coordinate, z. The three dimensional versions of the basic physical equations of (6)-(8) are

$$u_x + v_y + w_z = 0, \tag{77}$$

$$u_t + uu_x + vu_y + wu_z + p_x = (\sigma_{11})_x + (\sigma_{12})_y + (\sigma_{13})_z + fv, \tag{78}$$

$$v_t + uv_x + vv_y + wv_z + p_y = (\sigma_{11})_x + (\sigma_{22})_y + (\sigma_{13})_z - fu, \tag{79}$$

$$w_t + uw_x + vw_y + ww_z + p_z = (\sigma_{31})_x + (\sigma_{23})_y + (\sigma_{33})_z - g, \tag{80}$$

where density is a constant equal to one and v is the velocity in the direction of the new lateral coordinate y. In addition, the Coriolis acceleration is now an important factor; fv and fu represent its effect in this model. From this equation we proceed to a set of differential equations. Instead of having just one equation where we solve for $\frac{du}{dt}$ as a function of x and u, we obtain two equations; one for $\frac{du}{dt}$ and one for $\frac{dv}{dt}$, both in terms of x, y, u, and v.

The one important difference between the implementation of the two- and three-dimensional models is the "geometry"—the decomposition of the lateral space in the fluid domain into particle locations and fluid elements. This difference is discussed in greater detail in the next section. Another significant difference is that the three-dimensional model abandons the implicit Euler method for solving the differential equations in favor of the faster explicit Euler method. Because the number of particles tracked by the three-dimensional model is generally much larger than in the two-dimensional model, speed is of greater importance. In addition, this project's implementation of the three-dimensional model ignores some complicating factors incorporated into the two-dimensional model. This removes one of the chief advantages of the implicit method in the two-dimensional problem; namely, that the implicit method is more stable with complex or unusual initial conditions.

While the equations are being solved by a different method, the process for achieving a solution is still similar. The three-dimensional model still uses the same two-step operator

splitting method as the two-dimensional model. In the first part of each time step x and y are held constant and $\bar{\bar{u}}$ and $\bar{\bar{v}}$ are updated; in the second part $\bar{\bar{u}}$ and $\bar{\bar{v}}$ are held constant and x and y are updated. The three-dimensional model uses the same time grid as the two-dimensional; in both versions dt is the length of the time interval between time t^n and t^{n+1}. The two steps in the three-dimensional model can be described by

Step 1	**Step 2**
$t^n \leq t \leq t^{n+1}$	$t^n \leq t \leq t^{n+1}$
$\frac{d\mathbf{x}}{dt} = \frac{d\mathbf{y}}{dt} = 0$	$\frac{d\mathbf{x}}{dt} = \mathbf{u}$, $\frac{d\mathbf{y}}{dt} = \mathbf{v}$
$\mathbf{M_x}\frac{d\mathbf{u}}{dt} = \mathcal{F}(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v}),\ \mathbf{M_y}\frac{d\mathbf{v}}{dt} = \mathcal{G}(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v})$	$\mathbf{M_x}\frac{d\mathbf{u}}{dt} = \mathbf{M_y}\frac{d\mathbf{v}}{dt} = 0$

The mass matrices $\mathbf{M_x}$ and $\mathbf{M_y}$ differ from the mass matrix $\mathbf{M}$ of the two-dimensional model. The two-dimensional model used tri-diagonal matrices and required linear algebra to solve. Because we are trying to avoid the use of linear algebra, $\mathbf{M_x}$ and $\mathbf{M_y}$ are appropriately chosen diagonal matrices in the three-dimensional model.

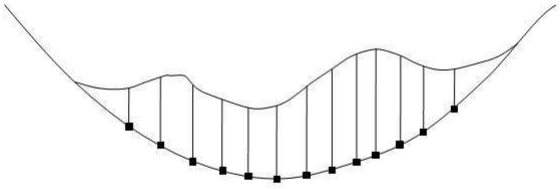

Figure 4: One-dimensional grid

9.1 The Geometry of the Three-Dimensional Model

A Lagrangian solution to the shallow water equations requires a grid where particles are identified at time zero and then tracked through all subsequent time steps. While the

two-dimensional model has a simple linear geometry, the three-dimensional model is much more complex. Since there is only one lateral degree of freedom in the two-dimensional model, the "grid" was simply a line with particle locations that could move back and forth across the grid in the one degree of freedom, as seen in (Figure 4). The $n+1$ particles divide this one-dimensional grid into n fluid elements.

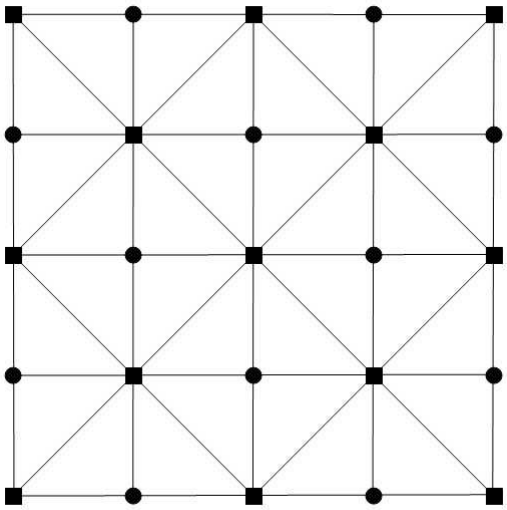

Figure 5: Two-dimensional grid

For the three-dimensional model, we use the two-dimensional grid depicted in (Figure 5). In this grid, there are two types of nodes which are both analogous to the particle locations of the two-dimensional model. Some nodes are connected to their four neighboring nodes in the four cardinal directions; these nodes are designated by circles in the figure. The other type of node is connected to eight neighbors–the four cardinal neighbors and four more diagonal neighbors–and is represented by a square. We cannot simply make every node an eight-node because then there would be additional points where edges intersected that were not in the initial node grid; specifically, where the diagonal edges of adjacent nodes crossed each other. The four- and eight-node grid allows for a greater number of edges while demanding that edges meet only at the nodes.

The significance of the edges is in the geometry of the triangles that they form in the grid. Each interior node of the grid has a square split into two triangles to its upper left. These two triangles are identified by the node to their lower right; the nodes are numbered sequentially as in (Figure 6). The triangles are divided into two classes depending on their spatial orientation, numbered either "1" or "2". Triangle numbers stay the same over vertical and horizontal edges lines, while alternating over diagonal edges. This is shown in (Figure

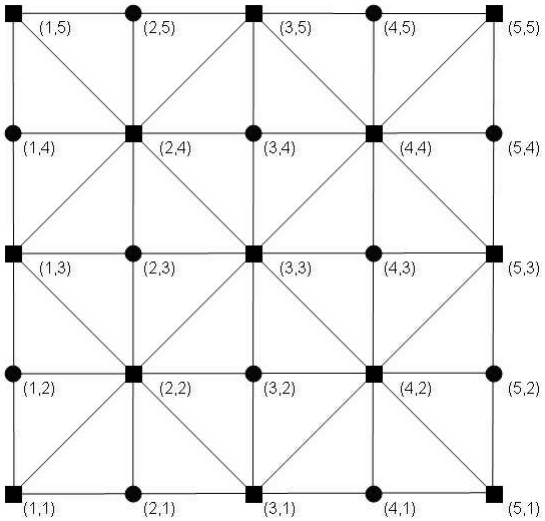

Figure 6: Numbered two-Dimensional grid

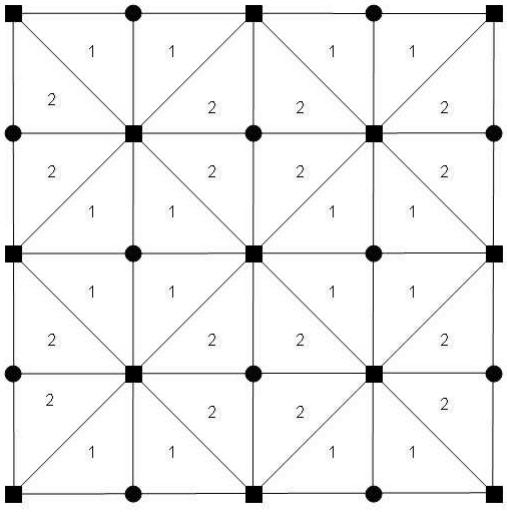

Figure 7: Two-dimensional grid with numbered triangles

7). We can see from this figure that each square has both a 1-type triangle and a 2-type triangle in it. This allows us to distinguish the two triangles from each other. Each triangle is identified by the node that is below it and on its left, as well as by its type number.

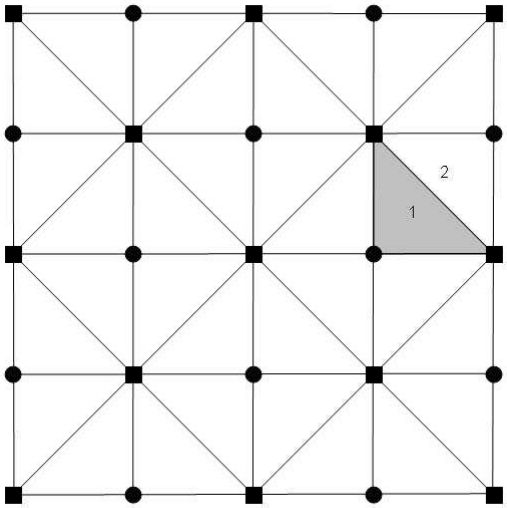

Figure 8: Two-dimensional grid with one pair of triangles highlighted

With this numbering scheme for triangles, we now have a way to label the edges of each triangle. For example, in (Figure 8), the darkened triangle can be identified as the "1-type" triangle belonging to node $(3, 4)$. When this model is implemented, it will be important to keep track of the changes in x, y, u, and v along each of the edges in the grid; this trianglular numbering scheme allows us to have a unique identifier for each edge. Another important piece of information to store is the area of each triangle. Because this is a Lagrangian scheme, the mass in each triangle at time zero will be constant for all time. In the two-dimensional model, this was also the case, and the water column in each fluid element was calculated by dividing the constant mass in that element—as established in the initial conditions at time zero—by the distance between the two fluid particles that form the boundary of that fluid element. In the three-dimensional version, the water column in each triangular element will be a constant that can be calculated from the constant mass in the triangle and the area of the triangle as it deforms through time.

The grid depicted in (Figure 5) has an eight-node—that is, a node that is connected to eight neighbors—at all four corners. If the nodes are numbered in a $m+1 \times n+1$ square, then both $m+1$ and $n+1$ must be odd in order for all corners to have an eight-node. The grid is implemented this way in order to be consistent, such that there will always be a set triangle geometry on the edges. By demanding a set geometry, the implementation of the

model becomes much easier: there is no longer any need to test the types of node at any given point. Instead we can be sure that all nodes where both m+1 and n+1 are both either odd or even are eight-nodes while all nodes where one of m+1 and n+1 is even and the other is odd are four-nodes. For example, $(1, 1)$ and $(4, 2)$ are both eight-nodes while $(1, 2)$ and $(4, 3)$ are four-nodes. It is important to know whether a node is an eight-node or four-node because that tells us about the orientation of the two triangles that belong to that node. For an eight-node, the hypotenuse of the two triangles has a positive slope, that is it goes from the bottom left of the square to the upper right. The hypotenuse of the four-node has a negative slope.

9.2 Implementing the Three-Dimensional Model in Serial

The serial version of the two-dimensional model solves the final form of the shallow water equations explicitly. This is done because the linear algebra needed to use the implicit Euler method becomes extremely time-consuming as the number of edges connecting nodes increases with the two-dimensional grid. Despite using the faster explicit Euler method to solve the differential equations of the operator splitting method, the three-dimensional model still runs significantly slower than the two-dimensional version. One reason is that more processor time is spent calculating the differences between nodes along the various edges; this data is needed to calculate pressure and viscosity terms. In the two-dimensional model there is a maximum of two edges per node along which differences must be calculated for the variables of interest: depth-averaged velocity, position, and water column. In the three-dimensional model, as noted in the previous section, there are up to eight edges per node; this is what causes the linear algebra to be impractical with the implicit Euler method. In addition, there are more variables of interest: depth velocity is represented by both a $\bar{\bar{u}}$ and $\bar{\bar{v}}$ component and position is represented by both x and y. The calculation of differences in water column is even more complex. In the two-dimensional version, the fluid elements are connected only at tracked particle locations, so it is only at these locations that we are interested in finding the difference in water columns. In the three-dimensional version the fluid elements are the triangles pictured in (Figure 5). There are approximately two triangles per node, and each triangle has three edges that have a unique water column difference.

Because the three-dimensional model uses the explicit method to solve its differential equations, care must be taken to ensure the stability of the model. In the two-dimensional model, the implicit method for numerical approximation was chosen to help ensure stability. Since the three-dimensional model uses the explicit method, it is necessary to choose time steps that are sufficiently small to ensure that there is no "cross-over." Cross-over is an error that occurs when the nodes of (Figure 5) move in such a way that some of the triangles overlap each other. The equivalent error in the two-dimensional model occurs when a particle that started with a x value greater than an adjacent particle gets a position value that is less than that adjacent node. Carefully choosing time-steps that are sufficiently small can prevent this error from occuring: in practice stability has been ensured when the ratio between dt

and the dx and dy of the initial grid is 0.02 or less.

The increased complexity of the three-dimensional model in serial causes greatly increased runtime; this is the motivation for the parallelization efforts of this project. The computationally intensive parts of the model must be organized in a way that makes it easier to divide the workload of the three-dimensional model over multiple processors. The serial implementation of the three-dimensional model can be broadly divided into three parts. Recall from the two-dimensional implementation that there are three lines of code which constitute the "core" of the model; these lines compute the depth-averaged velocity, position, and water column and exist within the time-stepping loop. The other code within the loop computes the various forces and matrices that are necessary to evaluate the three core lines. The three-dimensional code is constructed similarly. There are a few core lines that compute the three lines of interest, and these lines are inside the time-stepping loop along with code that calculate the necessary forces and matrices–which are much more complex in the three-dimensional model as mentioned above.

The three parts into which the three-dimensional model can be divided are everything inside of the time-stepping loop, the loop itself and everything outside the loop. The calculations inside the time stepping loop can be treated as a single-step calculation that takes the three variables of interest on a two-dimensional grid and returns these variables on that same grid one time step later. This can be treated as a separate function, and implementing the serial version in this way greatly eases the transition to the parallel version. Note that the grid input into this one-step function must follow all the rules established for the grid in the last section; specifically, that it must be rectangular and have an odd length in both dimensions in order to ensure that there is an eight-edge node on all four corners.

The loop itself is almost trivial in the serial version of the three-dimensional model. It only needs to incorporate the same two counting indices from the two-dimensional model—t and i—along with the logic for advancing the loop and ending it at the end of the runtime. The last part is the setup code that runs before entering the loop. This can also be treated as a separate self contained function.

9.3 Implementing the Three-Dimensional Model in Parallel

To produce efficient parallel code, work must be divided equally among several processors. A simple way to do this is to divide the two-dimensional particle grid evenly between the number of available processors. This then presents the problem of communication: each node has a separate set of differential equations to solve–specifically the solutions to

$$\mathbf{M_x}\frac{d\mathbf{u}}{dt} = \mathcal{F}(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$$

and

$$\mathbf{M_y}\frac{d\mathbf{v}}{dt} = \mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$$

which constitute the first step of the two-step operator splitting paradigm that this model uses. Both of these equations use vector variables to represent a set of $n + 1$ differential equations, each equation operating at one node on the two-dimensional grid. A sub-set of these equations and their associated nodes will be assigned to each separate processor. However, to solve any of the differential equations at a node, it is necessary to have data from all nodes connected to that node. If the node in question is on the edge of an area that has been split from the whole and assigned to a certain processor, then it may need data from nodes that have been assigned to a different subset on a different processor. Overcoming this problem requires a scheme for passing data between parallel processors, and this requirement drives the way in which the sub-domains are divided from the whole two-dimensional grid and assigned to processors.

Communication is, in general, more time consuming than processing. To optimize the parallel code, communication must be avoided as much as possible; therefore, the process for splitting the two-dimensional grid is designed to keep the amount of communication necessary small. Furthermore, each parallel processor should be assigned an equal share of the work to perform. Each time a processor pauses to communicate, it must wait for all processors to complete the computational tasks they had been assigned. Assuming that each processor is of roughly equal speed—an assumption that holds true for the Naval Academy's cluster—each processor should be assigned an equal sized sub-domain of the whole two-dimensional grid and equally sized sub-set of the differential equations.

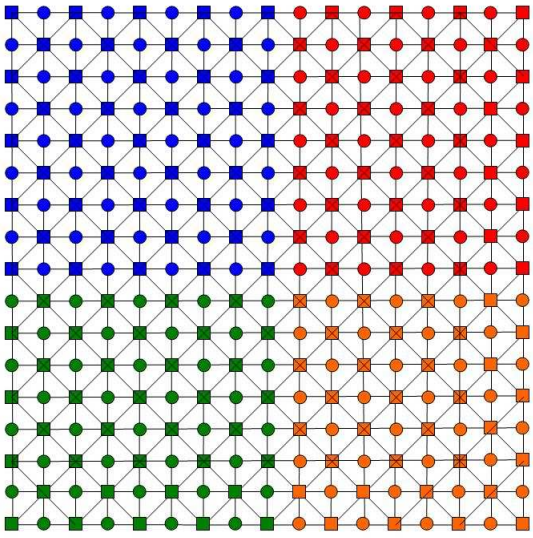

Figure 9: Two-dimensional grid divided into square sub-domains

Two of the easiest ways to divide the two-dimensional grid into sub-domains are by squares as depicted in (Figure 9) or by rectangles as in (Figure 10). If the goal is simply to

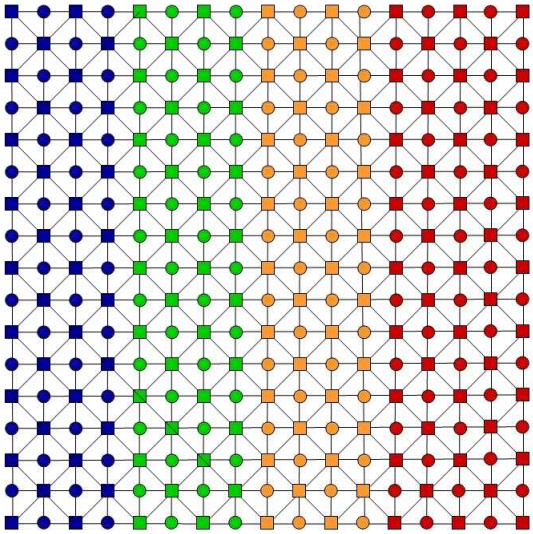

Figure 10: Two-Dimensional grid divided into rectangular sub-domains

divide in a way that results in minimal communication, the square is the superior method. For the rectangular method on an $m{+}1 \times n{+}1$ grid with P processors, there are $n+1\,(P-1)$ edges that cross sub-domain edges. For the same $m{+}1 \times n{+}1$ grid with sub-domains assigned to P processors by the square method there are $(m+n+2)\left(\sqrt{P}-1\right)$. Despite requiring more communication, the rectangular method has several advantages. Firstly, it is divided in only one coordinate. Thus, only the n columns in the x—or the y direction depending on the orientation of the grid—need be divided among the processors. This allows each processor to evaluate over the $1:m$ in the non-divided direction. This not only simplifies the code for solving differential equations at each node, but also greatly simplifies the code for communication between processors.

A second advantage of the rectangular method is in the pattern of communication. In a cluster using the square method for creating sub-domains, each processor may need to communicate with as many as eight other processors. With the rectangular method, each processor is guaranteed to only have two communication partners at most. Depending on the configuration of the hardware of the cluster, this can be a great advantage. For clusters organized in a lattice or a ring, the necessity to only communicate with two adjacent processors eliminates the need for any communication to pass through multiple processors in order to get from its source to its destination. Despite the fact that the Naval Academy's Beowulf cluster uses an InfiniBand Ethernet connection that allows all nodes to communicate with each other equally, this project's three-dimensional model uses the rectangular method to create sub-domains because of its increased viability on other types of clusters and especially because it greatly simplifies the process of writing code for communication between

processors. The rectangular shape of each sub-domain is the first constraint in the scheme for splitting the whole two-dimensional grid.

Determining that this project will use the rectangular method to create sub-domains from the whole two-dimensional grid sets the shape of each sub-domain. The next step is to determine the size of these sub-domains. The driving factor when determining the size is the three parts into which the serial implementation of the three-dimensional model was divided. Recall that all of the calculations—including the core code that computes the three variables of interest—within the time-stepping loop can be organized into a discrete single-step function. The parallel version of the code will use this same function completely unchanged from the serial version. In order to support this, the parallel version of the code must be able to pass this function data from the nodes of a two-dimensional grid that has all the same properties as the whole grid. The rectangular shape of each sub-domain already ensures that the m+1 dimension of each sub-domain is the same as in the whole m+1 $\times$ n+1 grid. The n+1 dimension of each sub-domain must have an odd number of nodes so that the four corner nodes of each sub-domain are eight-edge nodes. This is the second constraint for the sub-domains.

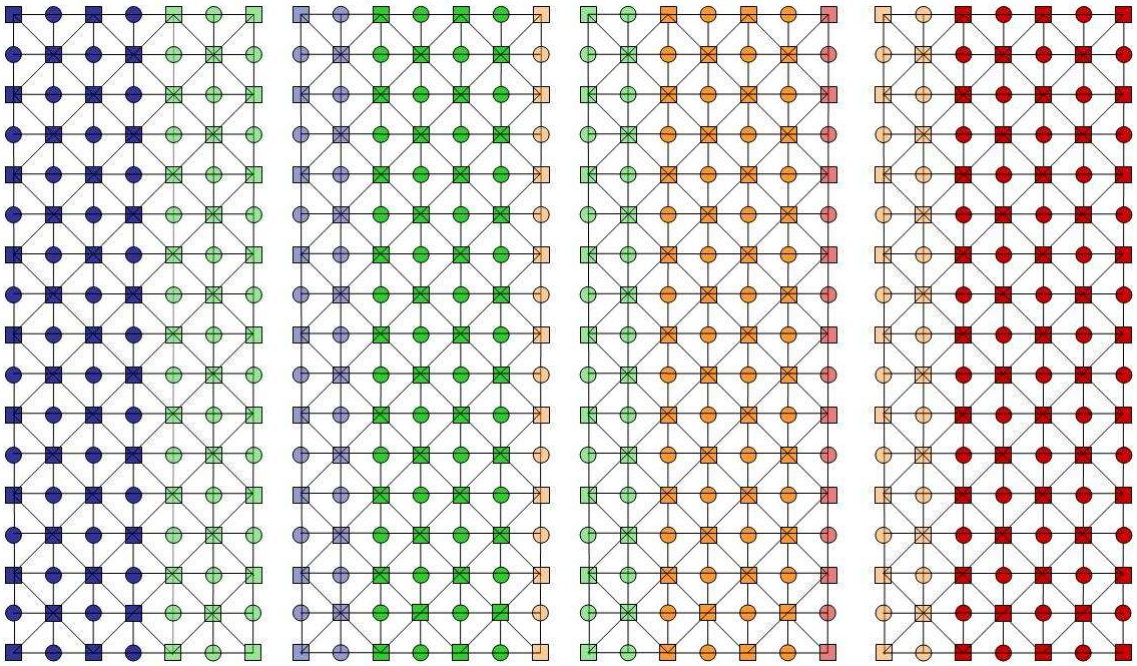

Figure 11: Sub-domain scheme with adjacent columns as used in three-dimensional model

A third constraint on each sub-domain is that each sub-domain should contain an extra column of nodes that "belong" to each of the up to two adjacent sub-domains. As

in (Figure 11), the left-most sub-domain should contain at least one column of nodes from the second left-most sub-domain. The reason for this is that in order for each node that "belongs" to a processor to be correctly updated, that processor must have access to the data from all adjacent nodes. These two constraints are enough to develop a scheme for splitting sub-domains from the whole two-dimensional grid.

Let the number of parallel processors that are available to the model be $P = 2^p$ and let the number of nodes along the $m + 1$ dimension of the two-dimensional grid be $m + 1 = 2^M + 1$. While these two assumptions hold, each processor will be responsible for a sub-domain of

$$\frac{2^M}{2^p} = 2^{M-p}$$

columns of $n+1$ nodes, except for the right most sub-domain which will "own" the one extra column for a total of $2^{M-p} + 1$ columns of $n + 1$ nodes. This division is pictured in (Figure 10). To fulfill the second constraint each sub-domain will have attached to it a column of nodes from each adjacent sub-domain. To fulfill the third constraint, each sub-domain will have one extra column from the adjacent sub-domain to the left appended to it as in (Figure 11). As can be seen in the figure, each of the sub-domains now has a rectangular shape, has eight-edge nodes at all four corners, and contains all the data necessary for the processor to perform single-step calculations on all the nodes that processor owns. Each processor has been assigned $2^{M-p} + 3$ nodes except for the first processor which only has $2^{M-p} + 1$. It would be better for each processor to be assigned the exact same number of nodes, thereby allowing each processor to run exactly the same single-step computation function. Therefore, the first processor is assigned two extra columns of nodes to the right of it, bringing its node count even with all the other processors.

The parallel version of the model first runs a modification of the set-up code that began the serial version. It then moves into the loop. During each iteration of the loop, it runs the same single-step function that was run in the serial version; making this possible is the prime motivation for the scheme for grid splitting. The last piece that must be added to make the parallel version work is a section within the loop that communicates the necessary data between processors. The geometry establishes that each processor needs to be passed only two columns of node data—or one column for the two end processors—to run each time step. Therefore all interior processors must pass two columns of node data and receive two columns of this data, while the end processors must pass and receive one column.

The extra columns of nodes that were appended to each sub-domain to comply with geometry requirements do not need to be passed. There were initially concerns that because this data is not updated to reflect the changes taking place in other parts of the model, that it might eventually become so divergent from the data that is passed that it would cause errors. The most likely error that would be caused would be the "zero area triangle" error in which one of the triangle's area approaches zero and therefore its water column approaches

infinity. However, it has been determined experimentally that although this data will not be updated accurately, it will remain "reasonable" enough to not cause any errors in the program; no run of the three-dimensional model has ever caused this error.

10 Results of Test Data and Conclusions

Table 1: Results of Testing

Domain Size	Serial	4 Proc. Rect.	4 Processor Parallel	8 Proc. Rect.	8 Processor Parallel	16 Proc. Rect.	16 Processor Parallel
65 x 65	0.7953	0.2839	0.3766	0.1939	0.2976	0.1461	0.3158
129 x 129	4.1803	0.8650	0.9338	0.4878	0.5743	0.3205	0.5210
257 x 257	24.5459	4.7274	5.0877	1.8032	1.9852	0.9316	1.1136

The data in (Table 1) was generated through test runs on the Naval Academy's Beowulf cluster with the MATLAB Distributed Computing Toolbox. The tests are run on square two-dimensional grids whose sides are of length $2^n + 1$. Three grid sizes were tested: $n = 6$, $n = 7$, and $n = 8$. The code for both the serial and parallel versions were modified from the versions attached in (D) and (E) to track runtime. Note that the times in (Table 1) are not processor times but "real" times. The code starts tracking time for each test after the initialization of data is done and right before the time-stepping loop begins. In the parallel version, this means that the initial grid and data are distributed to all processors before the timing starts.

The domain size is the number of nodes along each edge of a square grid created for the test. The column of data labeled "Serial" is that runtime for that domain size in the serial version. The data labeled "Parallel" is the runtime for P parallel processors. The data labeled "Rect." represents the runtime for one single processor when the domain is divided between P parallel processors. This represents the theoretical maximum speed of the model without any communication. The data in (Table 2) is calculated from the data in (Table 1) and shows the speedup factor for each domain size and number of processors, as well as the percentage of time spent communicating.

Table 2: Speedup and Communication Time

Domain Size	4 Processor Speedup	4 Processor Comm. Time	8 Processor Speedup	8 Processor Comm. Time	16 Processor Speedup	16 Processor Comm. Time
65 x 65	2.11	24.6%	2.67	34.8%	2.52	53.7%
129 x 129	4.48	7.4%	7.28	15.1%	8.02	38.5%
257 x 257	4.82	7.1%	12.36	9.2%	22.04	16.4%

The results of the test data show that the implementation of the three-dimensional model in parallel is successful. For a sufficiently large grid, increasing the number of processors available will increase the speed of the model. There is a reduction in efficiency as

the number of processors increases. This is result of increased communication between the processors. However, this is partially counterbalanced by a significant reduction in runtime for smaller domains. The ratio of runtime between larger and smaller domains is greater than the ratio between the number of nodes in each domain. This means that while one domain may be half the size of another, the processing time for the smaller domain will be less than half of the time for the larger domain. We suspect that this is because larger domains require more memory usage, and make proportionately less use of caching when performing calculations.

We see that the speedup on larger domains can exceed the number of processors. This can be seen in the tests on the grid of size n = 8, where the speedup with 8 processors is over 12, and the speedup with 16 processors is over 22. However, once the number of processors becomes too large, the losses introduced by communication overcome the advantages gained by using smaller domains. In the grid of size n = 6, 16 processors actually run slower than 8 processors because communication time takes up such a large percentage of the total runtime.

The results of this test data confirm that the methodology for splitting work between processors used in this project is a success. The deliverable three-dimensional model created by this Trident project is a proof-of-concept for the domain-splitting and communication strategy we introduce. As the complexity is added to the three-dimensional model we have created, the strategy for parallelization should remain valid.

References

[1] Blumberg, A., and Mellor, G. A description of a three-dimensional coastal ocean circulation model. In *Three-Dimensional Coastal Ocean Models*, N. S. Heaps, Ed. American Geophysical Union, Washington, DC, 1987, pp. 1–16.

[2] Gerbeau, J.-F., and Perthame, B. Derivation of viscous saint-venant system for laminar shallow water; numerical validation. *Discrete Continuous Dynamical Systems Series B 1*, 1 (2001), 89–102.

[3] Greenberg, J. M. Interim report on princeton ocean model assessment and modeling project. A report delivered to the Office of Naval Research regarding the status of Greenberg's work., 2007.

[4] Greenberg, J. M. Personal notes, 2007-2008.

[5] Kolmogorov, A. N. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Dokl. Akad. 30* (1941), 301. (English translation, Friedlander, S.K. and Topper L., *Turbulence*, Interscience, New York, 1961).

[6] LeVeque, R. J. *Numerical Methods for Conservation Laws*. Birkhauser Verlag, Basel, Switzerland, 1992.

[7] Malek-Madani, R. *Advanced Engineering Mathematics with Mathematica and MATLAB*, vol. 2. Addison-Wesley, Reading, Massachusetts, 1998.

[8] Mellor, G. User's guide for a three-dimensional, primitive equation, numerical ocean model (june 2003 version). In *Program in Atmospheric and Oceanic Science* (Princeton, New Jersey, June 2003), Princeton University, p. 53.

[9] Mellor, G. L. *Introduction to Physical Oceanography*. Princeton University, Princeton, New Jersey, 1996.

[10] Mellor, G. L., and Yamada, T. Development of a turbulence closure model for geophysical fluid problems. *Reviews of Geophysics and Space Physics 20*, 4 (1982), 851–875.

[11] Rotta, J. C. Turbulent shear layer prediction on the basis of the transport equations for the reynolds stresses. In *Proceedings of the 13th International Congress on Theoretical and Applied Mechanics, Moscow University* (New York, 1973), Springer-Verlag, pp. 295–308.

[12] Weiyan, T. *Shallow Water Hydrodynamics: Mathematical Theory and Numerical Solution for a Two-dimensional System of Shallow Water Equations*. Water and Power Press, Beijing, 1992.

[13] Yershov, A. *Numerical Methods for Shallow Water Equations.* PhD thesis, Carnegie Mellon University, 1998.

A Surface Traction Derivation

The normal to the surface $\bar{z} = \bar{a}(\bar{x})$ is

$$\mathbf{n}_{\bar{a}} = \frac{1}{\sqrt{1+\lambda^2\bar{a}_{\bar{x}}^2}} \begin{bmatrix} -\lambda\bar{a}_{\bar{x}} \\ 1 \end{bmatrix} \tag{A-1}$$

and the normal to the surface $\bar{z} = \bar{a}(\bar{x}) + \bar{h}(\bar{x},\bar{t})$ is

$$\mathbf{n}_{\bar{a}+\bar{h}} = \frac{1}{\sqrt{1+\lambda^2\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right)^2}} \begin{bmatrix} -\lambda\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right) \\ 1 \end{bmatrix} \tag{A-2}$$

Using the values of $\bar{\sigma}_{11}$ and $\bar{\sigma}_{13}$ from (17) and (A-1) and (A-2) we find that

$$\langle\bar{\sigma}_{11},\bar{\sigma}_{13}\rangle \cdot \mathbf{n}_{\bar{a}} = \frac{-2\epsilon\bar{a}_{\bar{x}}\bar{u}_{\bar{x}}(\bar{x},\bar{a},\bar{t}) + \epsilon\left(\bar{w}_{\bar{x}}(\bar{x},\bar{a},\bar{t}) + \frac{1}{\lambda^2}\bar{u}_{\bar{z}}(\bar{x},\bar{a},\bar{t})\right)}{\sqrt{1+\lambda^2\bar{a}_{\bar{x}}^2}} \tag{A-3}$$

and

$$\langle\bar{\sigma}_{11},\bar{\sigma}_{13}\rangle \cdot \mathbf{n}_{\bar{a}+\bar{h}} = \frac{-2\epsilon\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right)\bar{u}_{\bar{x}}(\bar{x},\bar{a}+\bar{h},\bar{t}) + \epsilon\left(\bar{w}_{\bar{x}}(\bar{x},\bar{a}+\bar{h},\bar{t}) + \frac{1}{\lambda^2}\bar{u}_{\bar{z}}(\bar{x},\bar{a}+\bar{h},\bar{t})\right)}{\sqrt{1+\lambda^2\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right)^2}} \tag{A-4}$$

Let $\hat{\mathbf{J}}$ be the physical Cartesian stress tensor

$$\hat{\mathbf{J}} = \begin{bmatrix} \hat{\sigma}_{11} & \hat{\sigma}_{13} \\ \hat{\sigma}_{31} & \hat{\sigma}_{33} \end{bmatrix}$$

or in non-dimensionalized form

$$\bar{\mathbf{J}} = \frac{EU}{l} \begin{bmatrix} 2\bar{u}_{\bar{x}} & \frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}} \\ \frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}} & 2\bar{w}_{\bar{z}} \end{bmatrix} \tag{A-5}$$

The traction vectors are

$$\mathbf{T}_{\bar{a}} = \bar{\mathbf{J}} \cdot \mathbf{n}_{\bar{a}}$$

and

$$\mathbf{T}_{\bar{a}+\bar{h}} = \bar{\mathbf{J}} \cdot \mathbf{n}_{\bar{a}+\bar{h}}.$$

We evaluate the traction vectors using (A-1), (A-2), and (A-5) to obtain

$$\mathbf{T}_{\bar{a}} = \frac{EU}{l} \begin{bmatrix} -2\lambda\bar{a}_{\bar{x}}\bar{u}_{\bar{x}} + \frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}} \\ \bar{u}_{\bar{z}} + \lambda^2\bar{w}_{\bar{x}} + 2\bar{w}_{\bar{z}} \end{bmatrix} \frac{1}{\sqrt{1+\lambda^2\bar{a}_{\bar{x}}^2}} \tag{A-6}$$

and

$$\mathbf{T}_{\bar{a}+\bar{h}} = \frac{EU}{l} \begin{bmatrix} -2\lambda\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right)\bar{u}_{\bar{x}} + \frac{1}{\lambda}\bar{u}_{\bar{z}} + \lambda\bar{w}_{\bar{x}} \\ \bar{u}_{\bar{z}} + \lambda^2\bar{w}_{\bar{x}} + 2\bar{w}_{\bar{z}} \end{bmatrix} \frac{1}{\sqrt{1+\lambda^2\left(\bar{a}_{\bar{x}}+\bar{h}_{\bar{x}}\right)^2}} \tag{A-7}$$

Note that the first components of $\mathbf{T}_{\bar{a}}$ and $\mathbf{T}_{\bar{a}+\bar{h}}$ are $\langle \bar{\sigma}_{11}, \bar{\sigma}_{13} \rangle \cdot \mathbf{n}_{\bar{a}}$ from (A-3) and $\langle \bar{\sigma}_{11}, \bar{\sigma}_{13} \rangle \cdot \mathbf{n}_{\bar{a}+\bar{h}}$ from (A-4), respectively. The tangential components of these traction vectors are found by the dot product of the traction vectors and a unit vector tangential to the surface as in

$$S_{\bar{a}} = \left(\bar{\mathbf{J}} \cdot \mathbf{n}_{\bar{a}}\right) \cdot \bar{\mathbf{t}}_{\bar{a}}, \quad S_{\bar{a}+\bar{h}} = \left(\bar{\mathbf{J}} \cdot \mathbf{n}_{\bar{a}+\bar{h}}\right) \cdot \bar{\mathbf{t}}_{\bar{a}+\bar{h}}.$$

The tangential components of the traction vectors (A-6) and (A-7) are

$$S_{\bar{a}} = \frac{EU}{l\left(1+\lambda^2\bar{a}^2\right)} \left[\frac{\bar{u}_{\bar{z}}}{\lambda}\left(1-\lambda^2\bar{a}^2\right) + \lambda\bar{w}_{\bar{x}}\left(1-\lambda^2\bar{a}^2\right)\right] \tag{A-8}$$

and

$$S_{\bar{a}+\bar{h}} = \frac{EU}{l\left(1+\lambda^2(\bar{a}^2+\bar{h}^2)\right)} \left[\frac{\bar{u}_{\bar{z}}}{\lambda}\left(1-\lambda^2(\bar{a}^2+\bar{h}^2)\right) + \lambda\bar{w}_{\bar{x}}\left(1-\lambda^2(\bar{a}^2+\bar{h}^2)\right)\right] \tag{A-9}$$

B MATLAB Code: **initialize2d.m**

```
function struc = initialize2d()

    %***************************USER INPUT AREA*****************************
    %NOTE: ALL USER INPUT FUNCTIONS MUST BE DESIGNED FOR MATRIX OPERATIONS.
    %THUS INSTEAD OF *, /, AND ^, THE INPUT SHOULD BE IN THE FORM .*, ./,
    %AND .^ UNLESS THE OPERATION IS BETWEEN A VARIABLE AND A SCALAR.

    %Settings for this simulation. A number of features can be turned on or
    %off. THese features are set below, with either 'true' for on, or
    %'false' for off.

    %Timing - If set to 'true', this run will measure the execution time
    %for the number of time steps specified in 'timesteps'. If 'false,'
    %timesteps is ignored and runtime is set by runtime variable below.
    timing = false;
    timesteps = 100;

    %Movie - If set to true, this run will be recorded as a movie.
    %name is the name of the file to which the movie will be stored
    mov = false;
    name = 'movie1.avi';

    %Residual - If set to 'false' only the depth averaged mean will be
    %used in calculations, and none of the variation at different depths
    %will be considered
    resid = true;

    %Friction - If set to 'false' then there will be no friction terms in
    %the calculations. Note that setting friction to 'false' automatically
    %sets the residual to 'false' as well. Also note that the upper and
    %lower friction constants can be turned off individually with the Kw
    %and Kf constants below.
    friction = true;

    % Set N or dx. if isN is true, then val is N, if isN is false, then val
    % is dx.
    isN = true;
    val = 100;

    % Set the function defining the bathymetry as a function of x
    % If bathymetry is an inline function, bath_type is true, if it is an
    % m-file function, then bath_type is false.
    bath_type = true;
    bathym = '(x.^2)/2';

    % Set the domain of the bathymetry. This domain should be the absolute
    % boundary of the problem. For now the only purpose of this is to set
    % axes for plotting.
    hi_b = 3.5;
    lo_b = -3.5;

    % Set the initial domain of the fluid
    hi_f = 2;
    lo_f = 1;

    % Set the function defining the initial water column as a function of x
    % If water column is an inline function, water_type is true, if it is
    % an m-file function, then water_type is false.
    water_type = true;
    water = '0*x+5/3';
```

```
% Set the function defining the initial velocity of the fluid as a
% function of x
% If initial velocity is an inline function, vel_type is true, if it is
% an m-file function, then vel_type is false.
vel_type = true;
vel = '0*x+0';

% Set constants for surface frictions. Kw is the upper surface friction
% and Kw is the lower surface friction. If these terms are set to zero,
% then there will be no effect from that friction source.
Kf = 0.002;
Kw = 0.0005;

% Set wind as a function of x and t FOR NOW JUST X
% If wind is an inline function, wind_type is true, if it is a m-file
% function, then wind_type is false.
wind_type = true;
wind = '0*x-1';

% Set the value of lambda
lambda = .001;

% Set the value of mu. Mu is the percentage between 0 and 1 of the
% damping to be used
mu = .8;

% Set the ratio dt/dx. A typical value is 0.1
rat = 0.1;

% Set the runtime for this model.
time = 100;



%************************ERROR CHECKING AREA*************************


%************************CALCULATION AREA*****************************

% Set the features to be turned on and off
struc.res = resid;
struc.fric = friction;

% Set the bathymetry and its domain.
if bath_type
    struc.bath = inline(bathym,'x');
else
    struc.bath = bathym;
end
struc.hi = hi_b;
struc.lo = lo_b;

% For a given N, calculate the x-array: constant mass, variable dx
if isN == true
    struc.n = val+1;
    if water_type
        wat = inline(water,'x');
    else
        wat = water;
    end
    el_mass = quad(wat, lo_f, hi_f, 1.e-10) / val;
    int_dom = lo_f:(hi_f-lo_f)/10000:hi_f;
    cum = cumtrapz(int_dom, feval(wat, int_dom));
```

```
    struc.X(1) = lo_f;
    j = 1;
    for i = 1:10000
        if cum(i) > j * el_mass;
            j = j + 1;
            struc.X(end+1) = int_dom(i);
        end
    end
    struc.X(end+1) = hi_f;

    approx_dx = (hi_f - lo_f) / val;
    struc.dt = rat * approx_dx;


% For a given dx, calculate the x-array: constant dx
else
    % Since not every dx will divide evenly into the domain, round off
    % N to the next highest whole number and recalculate dx.
    struc.n = ceiling((hi_f-lo_f) / val)
    struc.dx = (hi_f-lo_f) / (struc.N-1)
    struc.dt = rat * struc.dx;
    struc.X = lo_f:struc.dx:hi_f;
end

% Set the initial h vector. First evaluate 'water' at all points in the
% x-array, then find the midpoints between these values, leaving N-1
% datapoints in h.
for i = 1:length(struc.X)-1
    struc.M(i) = quad(wat, struc.X(i), struc.X(i+1), 1.e-10);
end

% Set the initial u vector. Evaluate 'vel' at all points in the
% x-array.
if vel_type
    struc.U = feval(inline(vel,'x'), struc.X);
else
    struc.U = feval(vel, struc.X);
end

% Set friction constants of the model.
struc.k_lo = Kf;
struc.k_hi = Kw;

% Set the wind function for the model
if wind_type
    struc.uw = inline(wind,'x');
else
    struc.uw = wind;
end

% Set the lambda value for the model
struc.lam = lambda;

% Set the mu value for the model
struc.moo = mu;

% Set the runtime of the model.
if timing
    struc.time = timesteps;
    struc.timing = true;
else
    struc.time = time;
    struc.timing = false;
end
```

```
%Determing if movie or not
struc.mov = mov;
if mov
    struc.name = name;
else
    struc.name = '';
end
```

C MATLAB Code: **shallow2d.m**

```
function out = shallow2d()

%close all other windows
close

%%Autoruns initialize.m to get setup data. Alternate option is to allow user
%to specify input file in function definition.
data = initialize2d();

%used to time execution of the program
if data.timing
    tic;
end

%Get initialization data from initialize.m
resid = data.res;
friction = data.fric;
bath = data.bath;
n = data.n;
dt = data.dt;
X = data.X;
M = data.M;
U = data.U;
hi_b = data.hi;
lo_b = data.lo;
kw = data.k_hi;
kf = data.k_lo;
wind = data.uw;
lambda = data.lam;
mu = data.moo;
runtime = data.time;

%Set a vector of the mass in each interval and total mass in problem
H = M./diff(X);

%%initialize time and counting indices to zero
t = 0;
i = 0;

% sundry variables set to zero: HH and XX for formatted display, xc and vc
% are centers of mass and momentum
XX = 0;
HH = 0;
XC = 0;
VC = 0;
T= .1/dt;
mov = 0;

% constants for plotting
%axes for mass vs. momentum
AX = -2:.1:2;
AY = AX*0;

%x and y for plotting potential function
X_POT = data.lo:.1:data.hi;
Y_POT = feval(bath, X_POT);


%% calculate (time independant) terms
% Mass matrix
MM = spdiags([[M 0]'/8,3*([0 M] + [M 0])'/8,[0 M]'/8],-1:1,n,n);
% Viscosity matrix
```

```
MMV = spdiags([[M 0]'/8,([0 M] + [M 0])'/-8,[0 M]'/8],-1:1,n,n);
% Total mass
mass = sum(MM * ones(size(1:n))');

%% Looping
%If timing then count by index numbers, otherwise count by elapsed time
if data.timing
    runtime = runtime * dt;
end

while t <  runtime

    % update time increment to t_n+1 and increment counting index
    t = t + dt;
    i = i + 1;


    % WE UPDATE U TERM BY TERM

%%MASS AND VISCOCITY MATRICES

    %MM has already been set

    MMVV = MMV * mu;

    %END MASS MATRIX

%%PRESSURE TERMS
    if resid == true
        %This section evaluates the integral of the square of the residual at
        %each point x. This becomes one of the pressure terms in the solution
        %for the depth averaged mean velocity.

        %These next calculations must be evaluated at x_i-1/2 and x_i+1/2 for each
        %point x_i. This means there will be n+1 points of evaluation where
        %n is the number of x points. For the two endpoints at x_1/2 and
        %x_n+1/2, we will estimate their value by x_1 and x_n,
        %respsectively.
        U_AVG = ([U(1) U] + [U U(end)])/2;

        %We set X_AVG to equal all the midpoints so that we can find the
        %wind function for those points.
        X_AVG = ([X(1) X] + [X X(end)])/2;

        UW_AVG = feval(wind, X_AVG);

        %At the two points beyond the boundary x values, h equals zero.
        H_EVAL = [0 H 0];

        %This is optimized code generated by MAPLE. SOL represents the
        %vector of solutions to this integral at each point x_i+1/2 and
        %x_i-1/2 multiplied by h.
        T1 = kw*H_EVAL;
        T2 = UW_AVG-U_AVG;
        T4 = H_EVAL.^2;
        T5 = kf*T4;
        t12 = kf^2;
        T18 = 1./(1 + kw*H_EVAL/3 + t12*T4/3 + kw*kf*T4.*H_EVAL/12);
        T20 = 1./(1 + T1/3 + t12*T4/3 + kw*kf*T4.*H_EVAL/12);
        T29 = T1.*T2.*T20;
        T35 = (1 + T1/4)*kf.*T4.*U_AVG.*T20;
        T37 = -T29/6 - T35/6;
        T40 = T1.*T2/2 - T5.*U_AVG/2 - T1.*((1/3 + T5/12)*kw.*H_EVAL.* ...
            T2.*T20 + T5.*U_AVG.*T20/6)/2 - T5.*T37/2;
```

```
        T41 = T40.^2;
        T45 = U_AVG - T29/6 - T35/6;
        T51 = T4.^2;
        T53 = T45.^2;
        T59 = T37.^2;
        SOL = H_EVAL.*(T41/5 + T5.*T45.*T40/2 +2/3*T37.*T40 + t12*T51.* ...
            T53/3 + T37*kf.*T4.*T45 + T59);

        %The value of the integral over x at each point x is the difference
        %between adjacent terms
        PRES = .5*(([H 0].^2 - [0 H].^2) + diff(SOL));
    else % resid = false case
        PRES = .5*(([H 0].^2 - [0 H].^2));
    end
    %END PRESSURE TERMS

%%POTENTIAL TERM
    BATH = feval(bath,X);   %bathymetry evaluated at each point in X
    BATH_MID = BATH(1:end-1) + diff(BATH)/2; %bathymetry at midpoints in X
    POT = [0 H] .* (BATH - [0 BATH_MID]) + [H 0] .* ([BATH_MID 0] - BATH);
    %END POTENTIAL TERM

%%FRICTION TERMS
    if friction == true

        %Evaluate wind at all X points from the user-defined function
        %passed to this method in wind.
        UW = feval(wind, X);

        %temporary solution: average h_i-1 and h_i to get the value at x_i
        H_AVG = ([0 H] + [H 0])/2;     %This is may not be correct

        %This section calculates the values of u_tilde if the residual
        %functionality is turned on by the user, otherwise it returns a
        %zero array for the residual values.
        if resid == true

            %This section calculates the values of u_tilde at s=0 and s=1.
            %These formulae were automatically generated by Maple.
            T1 = kw*H_AVG;
            T2 = UW-U;
            T4 = H_AVG.^2;
            T5 = kf*T4;
            t12 = kf^2;
            T18 = 1./(1 + kw*H_AVG/3 + t12*T4/3 + kw*kf*T4.*H_AVG/12);
            T20 = 1./(1 + T1/3 + t12*T4/3 + kw*kf*T4.*H_AVG/12);

            UHI = (1/3 + T5/12)*kw.*H_AVG.*T2.*T18 + T5.*U.*T18/6;
            ULO = -T1.*T2.*T20/6 - (1 + T1/4)*kf.*T4.*U.*T20/6;
        else  %resid = false case

            UHI = zeros(1, length(X));
            ULO = UHI;
        end

        %This section creates the matrices M_fr and M_w used to multiply
        %the UW, U, U(1), and U(0) vectors.

        %Using this method means that extended edges can cause way too much
        %friction.
        %DELTA_X = diff(X).^3

        %By making the DELTA_X constant, we prevent having runaway friction
        %at the edges.
```

```
        DELTA_X = ((X(end)-X(1)) / (n - 1))^3 * ones(1,n-1);

        MMWW = ((kw*mu)/(64*lambda^2))* spdiags([[DELTA_X 0]',...
            3*([0 DELTA_X] + [DELTA_X 0])',[0 DELTA_X]'],-1:1,n,n);

        DELTA_X_H = DELTA_X .* H;

        MMFF = ((kf*mu)/(64*lambda^2))* spdiags([[DELTA_X_H 0]',...
            3*([0 DELTA_X_H] + [DELTA_X_H 0])',[0 DELTA_X_H]'],-1:1,n,n);

        %The simpler method is to pick a constand to represent M_fr and m_w


    end
    %END FRICTION

%%solve for u
    %EXPLICIT, NO FRICTION
    %U = (MM\((MM + MMVV)*U' - dt*PRES' - dt*POT'))';

    if friction == false
        %IMPLICIT, NO FRICTION
        U = ((MM - MMVV)\(-dt*PRES' - dt*POT' + MM*U'))';
    else
        %IMPLICIT, WITH FRICTION
        U = ((MM - MMVV + dt* (MMWW + MMFF)) \ (-dt*POT' - dt*PRES' + ...
            dt*MMWW*(UW - UHI)' - dt*MMFF*ULO' + MM*U'))';
    end

    % update x at t_n+1
    X = X + U*dt;

    % calculate h at t_n+1
    H = M./diff(X);

%% Plotting
    if i-T*floor(i/T) < 1 | i == 1



        XX = X;

        HH = ([0 H] + [H 0])/2;
        HH(1) = 0;
        HH(end) = 0;

        %calculate center of mass and momentum in continuing vectors
        %not a general solution...depends specifically on the masses being
        %equally distributed among the intervals


        XC = [XC sum(MM*X')/mass];
        VC = [VC sum(MM*U')/mass];

        %set title fields
        ti = ['Water Elevation - red, Bottom Profile - black, Water' ...
            ' Height - blue vs. distance'];
        tstr = sprintf('Time: %.1f',t);
        istr = sprintf('Index: %d', i);
        indices = [tstr '                                                                ' istr];
        lstr = sprintf('Left Boundary: %.4f',X(1));
        cmass = sprintf('Center of Mass: %.4f', XC(end));
        rstr = sprintf('Right Boundary: %.4f',X(n));
        dists = [lstr '     ' cmass '      ' rstr];
```

```
        %make second version of H for elevation above potential
        H_PLOT = HH + feval(bath, XX);

        %this is the lower end of the plot. this needs to be user set
        %somehow
        lower = 0;


        %Alternate displays for movie or otherwise
        if data.timing
        else if data.mov %movie case
                mov = mov + 1

                plot(X_POT,Y_POT,'k',XX,HH+lower,'b',XX,H_PLOT,'r',X(1), ...
                    feval(bath,X(1)),'+ r',X(n),feval(bath,X(n)),'+ r');
                title(strvcat(ti, indices, dists));
                axis([lo_b hi_b 0 ((3.5)^2)/2]);

                if i == 1
                    set(1,'Position',[100,200,1000,400])
                    pause;
                    playme(mov) = getframe;

                else
                    playme(mov) = getframe;
                end
            else %not movie case

                %display height curves to first plot frame
                subplot(211);
                plot(X_POT,Y_POT,'k',XX,HH+lower,'b',XX,H_PLOT,'r',X(1), ...
                    feval(bath,X(1)),'+ r',X(n),feval(bath,X(n)),'+ r');
                title(strvcat(ti, indices, dists));
                axis([lo_b hi_b 0 ((3.5)^2)/2]);
                %axis([data.lo data.hi lower 6]);

                %display center of mass vs center of momentum plot
                subplot(212);
                plot(AX,AY,'k',AY,AX,'k',XC(2:end),VC(2:end),'b',XC(end), ...
                    VC(end),'+ b');
                title('Center of Mass vs. Average Momentum - blue');
                axis([-2 2 -2 2]);
                axis square

                drawnow
                if data.mov
                    playme(mov) = getframe;
                end
                if i == 1
                    pause;
                end
            end
        end

    end

end

if data.timing
    toc
end

if data.mov
```

```
    movie2avi(playme, data.name, 'fps', 10)
end


%%create the output data
if data.timing
    out = toc;
else
    out.res = resid;
    out.bath = bath;
    out.hi = hi_b;
    out.lo = lo_b;
    out.n = n;
    out.dt = dt;
    out.X = X;
    out.H = H;
    out.U = U;
    out.time = 0;
    out.k_hi = kw;
    out.k_lo = kf;
    out.uw = wind;
    out.lam = lambda;
    out.timing = data.timing;
    out.mov = data.mov;
    out.name = data.name;

end


end
```

D MATLAB Code: **initialize3d.m**

```
function data = initialize3d()


%% Data input for initial conditions

% 2^N+1 is the number of grid lines in each direction in the original
% square domain
N=5;

% Coefficient of rotation: analogous to a coriolis force
b=1.5;

% Maximum time before computation stops
Tmax=10;

% Bottom friction coefficient: a=0 corresponds to no friction
a = 0;

% Initial x-coordinate of the fluid center of mass
xc = 1/sqrt(2);

% Initial y-coordinate of the fluid center of mass
yc = 0;

% Eddy viscosity
mu= .05;

% Ratio of dt/dx
a1 = .01;

%% Setting up domain and initial values of X, Y, u, and v
m=2^N;

x=-.5:1/m:.5;
y=ones(size(1:m+1));

X=(y'*x)';
Y=(x'*y)';

dx=1/m ;
H=1;

dt=a1*dx;

d=0;
s=0;
w=0;
T=0;
u=((d+w)*X+(w+T)*Y)/2;
v=((T-w)*X+(d-s)*Y)/2;
X=X+xc;
Y=Y+yc;

%% Calculating area of the fluid elements

DxX1(1:m,1:2:m-1)=X(2:m+1,1:2:m-1)-X(1:m,1:2:m-1);
DxY1(1:m,1:2:m-1)=Y(2:m+1,1:2:m-1)-Y(1:m,1:2:m-1);
DxX1(1:m,2:2:m)=X(2:m+1,3:2:m+1)-X(1:m,3:2:m+1);
DxY1(1:m,2:2:m)=Y(2:m+1,3:2:m+1)-Y(1:m,3:2:m+1);

DxX2(1:m,1:2:m-1)=X(2:m+1,2:2:m)-X(1:m,2:2:m);
DxY2(1:m,1:2:m-1)=Y(2:m+1,2:2:m)-Y(1:m,2:2:m);
```

```
DxX2(1:m,2:2:m)=X(2:m+1,2:2:m)-X(1:m,2:2:m);
DxY2(1:m,2:2:m)=Y(2:m+1,2:2:m)-Y(1:m,2:2:m);

DyX1(2:2:m,1:m)=X(2:2:m,2:m+1)-X(2:2:m,1:m);
DyY1(2:2:m,1:m)=Y(2:2:m,2:m+1)-Y(2:2:m,1:m);
DyX1(1:2:m-1,1:m)=X(2:2:m,2:m+1)-X(2:2:m,1:m);
DyY1(1:2:m-1,1:m)=Y(2:2:m,2:m+1)-Y(2:2:m,1:m);

DyX2(1:2:m-1,1:m)=X(1:2:m-1,2:m+1)-X(1:2:m-1,1:m);
DyY2(1:2:m-1,1:m)=Y(1:2:m-1,2:m+1)-Y(1:2:m-1,1:m);
DyX2(2:2:m,1:m)=X(3:2:m+1,2:m+1)-X(3:2:m+1,1:m);
DyY2(2:2:m,1:m)=Y(3:2:m+1,2:m+1)-Y(3:2:m+1,1:m);


% area of type 1 and type 2 triangles assigned to each node
A1=(DxX1.*DyY1-DyX1.*DxY1)/2;
A2=(DxX2.*DyY2-DyX2.*DxY2)/2;


% mass in each fluid element
m1=H*A1;
m2=H*A2;


%%  Mass Matrix Computation

mass=sum(sum(sum(A1))+sum(sum(A2)))/m/m;
M=zeros(m+1);
M(1:m+1,1:m+1)=mass;

% Each processor calculates its own mass matrix
M(1,2:m)=mass/2;
M(m+1,2:m)=mass/2;
M(2:m,1)=mass/2;
M(2:m,m+1)=mass/2;
M(1,1)=mass/4;
M(m+1,m+1)=mass/4;
M(m+1,1)=mass/4;
M(1,m+1)=mass/4;

data.MM=sum(sum(M));

%% Constants for splitting to processors
p = 3;
P = 4;
n = 2^(N-p+1);


data.m = m;
data.n = n;
data.b = b;
data.a = a;
data.mu = mu;
data.dt = dt;
data.dx = dx;
data.Tmax = Tmax;
data.P = P;
data.xc = xc;
data.yc = yc;

data.X = X;
data.Y = Y;
data.u = u;
data.v = v;
```

```
data.M = M;
data.m1 = m1;
data.m2 = m2;

end
```

E MATLAB Code: **shallowPar3d.m**

```
function plotData = shallow_par3d()


%% Create a data input objects only on processor 1
if labindex == 1

    input = initialize3d();


    P = input.P;
    m = input.m;
    n = input.n;

    pass(P).m = m;
    % constant data
    for i = 1:P
        pass(i).m = m;
        pass(i).n = n+2;
        pass(i).b = input.b;
        pass(i).a = input.a;
        pass(i).mu = input.mu;
        pass(i).dt = input.dt;
        pass(i).dx = input.dx;
        pass(i).runtime = input.Tmax;
        pass(i).P = input.P;
        pass(i).xc = input.xc;
        pass(i).yc = input.yc;
    end

    % vertex data
    pass(1).X = input.X(:,1:n+3);
    pass(1).Y = input.Y(:,1:n+3);
    pass(1).u = input.u(:,1:n+3);
    pass(1).v = input.v(:,1:n+3);
    pass(1).M = input.M(:,1:n+3);

    for k = 2:input.P-1
        pass(k).X = input.X(:,(k-1)*n-1:k*n+1);
        pass(k).Y = input.Y(:,(k-1)*n-1:k*n+1);
        pass(k).u = input.u(:,(k-1)*n-1:k*n+1);
        pass(k).v = input.v(:,(k-1)*n-1:k*n+1);
        pass(k).M = input.M(:,(k-1)*n-1:k*n+1);
    end

    pass(P).X = input.X(:,(P-1)*n-1:m+1);
    pass(P).Y = input.Y(:,(P-1)*n-1:m+1);
    pass(P).u = input.u(:,(P-1)*n-1:m+1);
    pass(P).v = input.v(:,(P-1)*n-1:m+1);
    pass(P).M = input.M(:,(P-1)*n-1:m+1);

    % triangle data
    pass(1).m1 = input.m1(:,1:n+2);
    pass(1).m2 = input.m2(:,1:n+2);

    for k = 2:P-1
        pass(k).m1 = input.m1(:,(k-1)*n-1:k*n);
        pass(k).m2 = input.m2(:,(k-1)*n-1:k*n);
    end

    pass(P).m1 = input.m1(:,(P-1)*n-1:m);
    pass(P).m2 = input.m2(:,(P-1)*n-1:m);
```

```
end

%% End setup, start distribution to labs

if labindex > 1
    data = labReceive(1);
else   % in this case labindex == 1
    for i = 2:P
        labSend(pass(i), i);
    end
    data = pass(1);
end

%% Set output object for constants
if labindex == 1
    consts.M = input.M;
    consts.MM = input.MM;
    consts.xc = input.xc;
    consts.yc = input.yc;
    consts.m = input.m;
    consts.n = input.n;
    consts.plotpoints = input.Tmax*10; %this is not yet dynamic
else
    consts = 0;
end

%% Begin run on each individual processor

runtime = data.runtime;
dt = data.dt;
P = data.P;

step=.1/dt; %used to determine when to plot
k = 0;      %index
t = 0;      %time elapsed

%% Set up plotting variables

plotData(runtime*10).empty = 0; %this is also not yet dynamic

c = 0; %plot count

%% Running Loop
while t <= runtime

    k = k + 1;
    t = t + dt;

%% Computation
    data = spOneStep(data);

%% Communication with neighbors

    % Share data with adjacent processors
    n = data.n;

    if labindex == 1
        labSend([data.X(:,n-3:n-2) data.Y(:,n-3:n-2) data.u(:,n-3:n-2) ...
            data.v(:,n-3:n-2)],2);
    else if labindex == P
            fromLeft = labReceive(P - 1);
        else
            fromLeft = labSendReceive(labindex + 1, labindex - 1, ...
                [data.X(:,n-1:n) data.Y(:,n-1:n) data.u(:,n-1:n) ...
```

```
                data.v(:,n-1:n)]);
        end
    end

    if labindex == 1
        fromRight = labReceive(labindex + 1);
    else if labindex == P
            labSend([data.X(:,3) data.Y(:,3) data.u(:,3) data.v(:,3)],...
                P-1);
        else if labindex == 2
                fromRight = labSendReceive(1, 3, ...
                    [data.X(:,3:5) data.Y(:,3:5) data.u(:,3:5) ...
                    data.v(:,3:5)]);
            else
                fromRight = labSendReceive(labindex - 1, labindex + 1, ...
                    [data.X(:,3) data.Y(:,3) data.u(:,3) data.v(:,3)]);
            end
        end
    end

    % Update matrices with data from adjacent processors

    if labindex ~= 1
        data.X(:, 1) = fromLeft(:,1);
        data.Y(:, 1) = fromLeft(:,3);
        data.u(:, 1) = fromLeft(:,5);
        data.v(:, 1) = fromLeft(:,7);
        data.X(:, 2) = fromLeft(:,2);
        data.Y(:, 2) = fromLeft(:,4);
        data.u(:, 2) = fromLeft(:,6);
        data.v(:, 2) = fromLeft(:,8);
    end
    if labindex ~= P
        if labindex == 1
            data.X(:, n-1) = fromRight(:,1);
            data.Y(:, n-1) = fromRight(:,4);
            data.u(:, n-1) = fromRight(:,7);
            data.v(:, n-1) = fromRight(:,10);
            data.X(:, n) = fromRight(:,2);
            data.Y(:, n) = fromRight(:,5);
            data.u(:, n) = fromRight(:,8);
            data.v(:, n) = fromRight(:,11);
            data.X(:, n+1) = fromRight(:,3);
            data.Y(:, n+1) = fromRight(:,6);
            data.u(:, n+1) = fromRight(:,9);
            data.v(:, n+1) = fromRight(:,12);
        else
            data.X(:, n+1) = fromRight(:,1);
            data.Y(:, n+1) = fromRight(:,2);
            data.u(:, n+1) = fromRight(:,3);
            data.v(:, n+1) = fromRight(:,4);
        end
    end

%% Plotting every 1/10 second

    jj=k-step*floor(k/step);

    if jj == 0

        c = c + 1

        if labindex == 1
```

```
            plotData(c).X = data.X(:,1:end-3);
            plotData(c).Y = data.Y(:,1:end-3);
            plotData(c).u = data.u(:,1:end-3);
            plotData(c).v = data.v(:,1:end-3);

            plotData(c).A1 = data.A1(:,1:end-3);
            plotData(c).A2 = data.A2(:,1:end-3);

        else if labindex == P

                plotData(c).X = data.X(:,3:end);
                plotData(c).Y = data.Y(:,3:end);
                plotData(c).u = data.u(:,3:end);
                plotData(c).v = data.v(:,3:end);

                plotData(c).A1 = data.A1(:,2:end);
                plotData(c).A2 = data.A2(:,2:end);

            else

                plotData(c).X = data.X(:,3:end-1);
                plotData(c).Y = data.Y(:,3:end-1);
                plotData(c).u = data.u(:,3:end-1);
                plotData(c).v = data.v(:,3:end-1);

                plotData(c).A1 = data.A1(:,2:end-1);
                plotData(c).A2 = data.A2(:,2:end-1);

            end
        end

        plotData(c).t = t;

    end

%% Loop

%% Assign objects into 'base' workspace for easy transfer to client
    assignin('base', 'out', plotData);
    assignin('base', 'static', consts);


end
```

F MATLAB Code: `spOneStep.m`

```
function data = spOneStep(data)

%% Data in from last step

% This data is constant
m = data.m; % number of grid lines in x-direction
n = data.n; % number of grid lines in y-direction
b = data.b; % rotation coefficient
a = data.a; % bottom friction coefficient
mu = data.mu; % eddy viscocity
dt = data.dt; % time step
%dx = data.dx;

m1 = data.m1; % matrices of masses
m2 = data.m2;
M = data.M;

% This data is changed
X = data.X; % matrix of x-coordinate values
Y = data.Y; % matrix of y-coordinate values
u = data.u; % matrix of x-components of velocity
v = data.v; % matrix of y-components of velocity


%% Calculate properties along edges of triangles
% positions differences

DxX1(1:m,1:2:n-1)=X(2:m+1,1:2:n-1)-X(1:m,1:2:n-1);
DxY1(1:m,1:2:n-1)=Y(2:m+1,1:2:n-1)-Y(1:m,1:2:n-1);
DxX1(1:m,2:2:n)=X(2:m+1,3:2:n+1)-X(1:m,3:2:n+1);
DxY1(1:m,2:2:n)=Y(2:m+1,3:2:n+1)-Y(1:m,3:2:n+1);

DxX2(1:m,1:2:n-1)=X(2:m+1,2:2:n)-X(1:m,2:2:n);
DxY2(1:m,1:2:n-1)=Y(2:m+1,2:2:n)-Y(1:m,2:2:n);
DxX2(1:m,2:2:n)=X(2:m+1,2:2:n)-X(1:m,2:2:n);
DxY2(1:m,2:2:n)=Y(2:m+1,2:2:n)-Y(1:m,2:2:n);

DyX1(2:2:m,1:n)=X(2:2:m,2:n+1)-X(2:2:m,1:n);
DyY1(2:2:m,1:n)=Y(2:2:m,2:n+1)-Y(2:2:m,1:n);
DyX1(1:2:m-1,1:n)=X(2:2:m,2:n+1)-X(2:2:m,1:n);
DyY1(1:2:m-1,1:n)=Y(2:2:m,2:n+1)-Y(2:2:m,1:n);

DyX2(1:2:m-1,1:n)=X(1:2:m-1,2:n+1)-X(1:2:m-1,1:n);
DyY2(1:2:m-1,1:n)=Y(1:2:m-1,2:n+1)-Y(1:2:m-1,1:n);

DyX2(2:2:m,1:n)=X(3:2:m+1,2:n+1)-X(3:2:m+1,1:n);
DyY2(2:2:m,1:n)=Y(3:2:m+1,2:n+1)-Y(3:2:m+1,1:n);

% calculate this data to avoid storing it

A1=(DxX1.*DyY1-DyX1.*DxY1)/2;
A2=(DxX2.*DyY2-DyX2.*DxY2)/2;

h1=m1./A1;
h2=m2./A2;

% velocity differences
Dxu1(1:m,1:2:n-1)=u(2:m+1,1:2:n-1)-u(1:m,1:2:n-1);
Dxv1(1:m,1:2:n-1)=v(2:m+1,1:2:n-1)-v(1:m,1:2:n-1);
Dxu1(1:m,2:2:n)=u(2:m+1,3:2:n+1)-u(1:m,3:2:n+1);
Dxv1(1:m,2:2:n)=v(2:m+1,3:2:n+1)-v(1:m,3:2:n+1);
```

```
Dxu2(1:m,1:2:n-1)=u(2:m+1,2:2:n)-u(1:m,2:2:n);
Dxv2(1:m,1:2:n-1)=v(2:m+1,2:2:n)-v(1:m,2:2:n);
Dxu2(1:m,2:2:n)=u(2:m+1,2:2:n)-u(1:m,2:2:n);
Dxv2(1:m,2:2:n)=v(2:m+1,2:2:n)-v(1:m,2:2:n);

Dyu1(2:2:m,1:n)=u(2:2:m,2:n+1)-u(2:2:m,1:n);
Dyv1(2:2:m,1:n)=v(2:2:m,2:n+1)-v(2:2:m,1:n);
Dyu1(1:2:m-1,1:n)=u(2:2:m,2:n+1)-u(2:2:m,1:n);
Dyv1(1:2:m-1,1:n)=v(2:2:m,2:n+1)-v(2:2:m,1:n);

Dyu2(1:2:m-1,1:n)=u(1:2:m-1,2:n+1)-u(1:2:m-1,1:n);
Dyv2(1:2:m-1,1:n)=v(1:2:m-1,2:n+1)-v(1:2:m-1,1:n);

Dyu2(2:2:m,1:n)=u(3:2:m+1,2:n+1)-u(3:2:m+1,1:n);
Dyv2(2:2:m,1:n)=v(3:2:m+1,2:n+1)-v(3:2:m+1,1:n);

%% Pressure terms in x momentum eqn.

p1RLx=(h1.^2).*(DyY1)/4;
p2RLx=(h2.^2).*(DyY2)/4;
p1UDx=(h1.^2).*(DxY1)/4;
p2UDx=(h2.^2).*(DxY2)/4;

P1rxx=0*X;
P1rxx(1:m,1:n)=p1RLx;

P1Lxx=0*X;
P1Lxx(2:m+1,1:n)=p1RLx;

P2rxx=0*X;
P2rxx(1:m,1:n)=p2RLx;

P2Lxx=0*X;
P2Lxx(2:m+1,1:n)=p2RLx;

P2Uxy=0*Y;
P2Uxy(1:m,1:n)=p2UDx;

P2Dxy=0*Y;
P2Dxy(1:m,2:n+1)=p2UDx;

P1Uxy=0*Y;
P1Uxy(1:m,1:n)=p1UDx;

P1Dxy=0*Y;
P1Dxy(1:m,2:n+1) = p1UDx;

% Pressure Forces in x eqn

PPx=0*X;
PPx(:,1)=PPx(:,1)+ dt*(P1Lxx(:,1)-P1rxx(:,1));
PPx(:,3:2:n+1)=PPx(:,3:2:n+1)+dt*( P1Lxx(:,3:2:n+1)-P1rxx(:,3:2:n+1));
PPx(:,3:2:n+1)=PPx(:,3:2:n+1)+dt*(P1Lxx(:,2:2:n)-P1rxx(:,2:2:n));
PPx(:,2:2:n)=PPx(:,2:2:n)+dt*(P2Lxx(:,1:2:n-1)-P2rxx(:,1:2:n-1));
PPx(:,2:2:n)=PPx(:,2:2:n)+dt*(P2Lxx(:,2:2:n)-P2rxx(:,2:2:n));
PPx(1,:)=PPx(1,:)-dt*(P2Dxy(1,:)-P2Uxy(1,:));
PPx(3:2:m+1,:)=PPx(3:2:m+1,:)-dt*(P2Dxy(3:2:m+1,:)-P2Uxy(3:2:m+1,:));
PPx(3:2:m+1,:)=PPx(3:2:m+1,:)-dt*(P2Dxy(2:2:m,:)-P2Uxy(2:2:m,:));
PPx(2:2:m,:)=PPx(2:2:m,:)-dt*(P1Dxy(1:2:m-1,:)-P1Uxy(1:2:m-1,:));
PPx(2:2:m,:)=PPx(2:2:m,:)-dt*(P1Dxy(2:2:m,:)-P1Uxy(2:2:m,:));


%% Pressure terms in y momentum eqn.
```

```
p2udy=(h2.^2).*(DxX2)/4;
p1udy=(h1.^2).*(DxX1)/4;
p2RLy=(h2.^2).*(DyX2)/4;
p1RLy=(h1.^2).*(DyX1)/4;

P2uyy=0*Y;
P2uyy(1:m,1:n)= p2udy;

P2dyy=0*Y;
P2dyy(1:m,2:n+1)=p2udy;

P1uyy=0*Y;
P1uyy(1:m,1:n)= p1udy;

P1dyy=0*Y;
P1dyy(1:m,2:n+1)= p1udy;


P1ryx=0*X;
P1ryx(1:m,1:n)= p1RLy ;

P1Lyx=0*X;
P1Lyx(2:m+1,1:n)=p1RLy;

P2ryx=0*X;
P2ryx(1:m,1:n)=p2RLy;

P2Lyx=0*X;
P2Lyx(2:m+1,1:n)= p2RLy;


% Pressure Forces in y eqn

PPy=0*Y;
PPy(1,:)=PPy(1,:)+dt*(P2dyy(1,:)-P2uyy(1,:));

PPy(3:2:m+1,:)=PPy(3:2:m+1,:)+dt*(P2dyy(3:2:m+1,:)-P2uyy(3:2:m+1,:));
PPy(3:2:m+1,:)=PPy(3:2:m+1,:)+dt*(P2dyy(2:2:m,:)-P2uyy(2:2:m,:));

PPy(2:2:m,:)=PPy(2:2:m,:)+dt*(P1dyy(1:2:m-1,:)-P1uyy(1:2:m-1,:));
PPy(2:2:m,:)=PPy(2:2:m,:)+dt*(P1dyy(2:2:m,:)-P1uyy(2:2:m,:));

PPy(:,1)=PPy(:,1)-dt*(P1Lyx(:,1)-P1ryx(:,1));

PPy(:,3:2:n+1)=PPy(:,3:2:n+1)-dt*(P1Lyx(:,3:2:n+1)-P1ryx(:,3:2:n+1));
PPy(:,3:2:n+1)=PPy(:,3:2:n+1)-dt*(P1Lyx(:,2:2:n)-P1ryx(:,2:2:n));

PPy(:,2:2:n)=PPy(:,2:2:n)-dt*(P2Lyx(:,2:2:n)-P2ryx(:,2:2:n));
PPy(:,2:2:n)=PPy(:,2:2:n)-dt*(P2Lyx(:,1:2:n-1)-P2ryx(:,1:2:n-1));



% this ends the pressure computation

%% viscous stresses computation

s1=mu*(h1).*(-(DxX1).*(Dyv1)+(Dxu1).*(DyY1)+(DyX1).*(Dxv1)-(DxY1).*(Dyu1));
s2=mu*(h2).*(-(DxX2).*(Dyv2)+(Dxu2).*(DyY2)+(DyX2).*(Dxv2)-(DxY2).*(Dyu2));

T1=mu*(h1).*((DxX1).*(Dyu1)+(DyY1).*(Dxv1)-(DyX1).*(Dxu1)-(DxY1).*(Dyv1));
T2=mu*(h2).*((DxX2).*(Dyu2)+(DyY2).*(Dxv2)-(DyX2).*(Dxu2)-(DxY2).*(Dyv2));

V1rxx=0*X;
V1rxx(1:m,1:n)=(s1.*DyY1-T1.*DyX1)/2;
```

```
V1Lxx=0*X;
V1Lxx(2:m+1,1:n)=(s1.*DyY1-T1.*DyX1)/2;


V2rxx=0*X;
V2rxx(1:m,1:n)=(s2.*DyY2-T2.*DyX2)/2;

V2Lxx=0*X;
V2Lxx(2:m+1,1:n)=(s2.*DyY2-T2.*DyX2)/2 ;


V1uxy=0*X;
V1uxy(1:m,1:n)=(T1.*DxX1-s1.*DxY1)/2;

V1dxy=0*X;
V1dxy(1:m,2:n+1)=(T1.*DxX1-s1.*DxY1)/2;

V2uxy=0*X;
V2uxy(1:m,1:n)=(T2.*DxX2-s2.*DxY2)/2;

V2dxy=0*X;
V2dxy(1:m,2:n+1)=(T2.*DxX2-s2.*DxY2)/2;



% Viscous Forces in x eqn
VVx=0*X;

VVx(:,1)=VVx(:,1)+(V1rxx(:,1)-V1Lxx(:,1));

VVx(:,3:2:n+1)=VVx(:,3:2:n+1)+(V1rxx(:,3:2:n+1)-V1Lxx(:,3:2:n+1));
VVx(:,3:2:n+1)=VVx(:,3:2:n+1)+(V1rxx(:,2:2:n)-V1Lxx(:,2:2:n));

VVx(:,2:2:n)=VVx(:,2:2:n)+ (V2rxx(:,2:2:n)-V2Lxx(:,2:2:n));
VVx(:,2:2:n)=VVx(:,2:2:n)+ (V2rxx(:,1:2:n-1)-V2Lxx(:,1:2:n-1));

VVx(1,:)=VVx(1,:)+(V2uxy(1,:)-V2dxy(1,:));

VVx(2:2:m,:)=VVx(2:2:m,:)+(V1uxy(2:2:m,:)-V1dxy(2:2:m,:));
VVx(2:2:m,:)=VVx(2:2:m,:)+(V1uxy(1:2:m-1,:)-V1dxy(1:2:m-1,:));

VVx(3:2:m+1,:)=VVx(3:2:m+1,:)+(V2uxy(3:2:m+1,:)-V2dxy(3:2:m+1,:));
VVx(3:2:m+1,:)=VVx(3:2:m+1,:)+(V2uxy(2:2:m,:)-V2dxy(2:2:m,:));



V1ryx=0*X;
V1ryx(1:m,1:n)=(T1.*DyY1+s1.*DyX1)/2;

V1Lyx=0*X;
V1Lyx(2:m+1,1:n)=(T1.*DyY1+s1.*DyX1)/2;

V2ryx=0*X;
V2ryx(1:m,1:n)=(T2.*DyY2+s2.*DyX2)/2;

V2Lyx=0*X;
V2Lyx(2:m+1,1:n)=(T2.*DyY2+s2.*DyX2)/2;


V1uyy=0*X;
V1uyy(1:m,1:n)=(T1.*DxY1+s1.*DxX1)/2;

V1dyy=0*X;
```

```
V1dyy(1:m,2:n+1)=(T1.*DxY1+s1.*DxX1)/2;

V2uyy=0*Y;
V2uyy(1:m,1:n)=(T2.*DxY2+s2.*DxX2)/2;

V2dyy=0*X;
V2dyy(1:m,2:n+1)=(T2.*DxY2+s2.*DxX2)/2;


% Viscous Forces in y eqn
VVy=0*Y;

VVy(:,1)=VVy(:,1)+(V1ryx(:,1)-V1Lyx(:,1));

VVy(:,3:2:n+1)=VVy(:,3:2:n+1)+(V1ryx(:,3:2:n+1) -V1Lyx(:,3:2:n+1));
VVy(:,3:2:n+1)=VVy(:,3:2:n+1)+(V1ryx(:,2:2:n)-V1Lyx(:,2:2:n));

VVy(:,2:2:n)=VVy(:,2:2:n)+(V2ryx(:,2:2:n)-V2Lyx(:,2:2:n));
VVy(:,2:2:n)=VVy(:,2:2:n)+(V2ryx(:,1:2:n-1)-V2Lyx(:,1:2:n-1));

VVy(1,:)=VVy(1,:)-(V2uyy(1,:)-V2dyy(1,:));

VVy(3:2:m+1,:)=VVy(3:2:m+1,:)-(V2uyy(3:2:m+1,:)-V2dyy(3:2:m+1,:));
VVy(3:2:m+1,:)=VVy(3:2:m+1,:)-(V2uyy(2:2:m,:)-V2dyy(2:2:m,:));

VVy(2:2:m,:)=VVy(2:2:m,:)-(V1uyy(2:2:m,:)-V1dyy(2:2:m,:));
VVy(2:2:m,:)=VVy(2:2:m,:)-(V1uyy(1:2:m-1,:)-V1dyy(1:2:m-1,:));

% this concludes viscous stress computation

%%  Computation of x and y Forces

Fx=PPx+VVx;
Fy=PPy+VVy;


% this concludes the force computation

%%  Velocity update

u=(1-a*dt)*u+(Fx./M-dt*(X-b*v));
v=(1-a*dt)*v+(Fy./M-dt*(Y+b*u));


%% Leap Frog

X=X+dt*u;
Y=Y+dt*v;


%% Store data to pass to next step
%update some data
A1=(DxX1.*DyY1-DyX1.*DxY1)/2;
A2=(DxX2.*DyY2-DyX2.*DxY2)/2;


%  only store data that has been changed
data.X = X; % matrix of x-coordinate values
data.Y = Y; % matrix of y-coordinate values
data.u = u; % matrix of x-components of velocity
data.v = v; % matrix of y-components of velocity

data.A1 = A1;
data.A2 = A2;
```

```
% This data is stored for plotting - DEPRECATED FOR NOW
%SS = s1.*s1+T1.*T1+s2.^2 +T2.^2;
%data.SS1_1 = max(max(SS));
%data.SS1_2 = max(max(max(A1)),max(max(A2)))/dx/dx;
%data.SS1_3 = min(min(A2))/dx/dx;

end
```

G MATLAB Code: `spPlot.m`

```
function spPlot()

%% Load data from labs to client
pmode lab2client out 1 set1
pmode lab2client out 2 set2
pmode lab2client out 3 set3
pmode lab2client out 4 set4
pmode lab2client static 1 static

%% Move data from base workspace to function workspace
static = evalin('base', 'static');
set1 = evalin('base', 'set1');
set2 = evalin('base', 'set2');
set3 = evalin('base', 'set3');
set4 = evalin('base', 'set4');

%% Extract data from structures
xc = static.xc;
yc = static.yc;
Time = 0;
R = xc^2+yc^2;

m = static.m;
n = static.m;
M = static.M;
MM = static.MM;

plotpoints = static.plotpoints;

[pData(1:plotpoints).empty] = deal(0);

for i = 1:plotpoints
    pData(i).X = [set1(i).X set2(i).X set3(i).X set4(i).X];
    pData(i).Y = [set1(i).Y set2(i).Y set3(i).Y set4(i).Y];
    pData(i).u = [set1(i).u set2(i).u set3(i).u set4(i).u];
    pData(i).v = [set1(i).v set2(i).v set3(i).v set4(i).v];

    pData(i).A1 = [set1(i).A1 set2(i).A1 set3(i).A1 set4(i).A1];
    pData(i).A2 = [set1(i).A2 set2(i).A2 set3(i).A2 set4(i).A2];

    pData(i).t = set1(i).t;

end

%% Plot
for c = 1:plotpoints

    X = pData(c).X; % matrix of x-coordinate values
    Y = pData(c).Y; % matrix of y-coordinate values
    u = pData(c).u; % matrix of x-components of velocity
    v = pData(c).v; % matrix of y-components of velocity

    A1 = pData(c).A1;
    A2 = pData(c).A2;

    t = pData(c).t;

    HH=0*X;
    AA=0*X;

    AA(2:2:m,2:2:n)=(A2(2:2:m,2:2:n)+A2(2:2:m,1:2:n-1)+A2(1:2:m-1,2:2:n)+...
                                A2(1:2:m-1,1:2:n-1))/4;
```

```
AA(2:2:m,2:2:n)=AA(2:2:m,2:2:n)+(A1(2:2:m,2:2:n)+A1(2:2:m,1:2:n-1)+...
                                A1(1:2:m-1,2:2:n)+...A1(1:2:m-1,1:2:n-1))/4;
AA(2:2:m,3:2:n-1)=(A1(2:2:m,3:2:n-1)+A1(1:2:m-1,3:2:n-1)+...
                                A1(1:2:m-1,2:2:n-2)+A1(2:2:m,2:2:n-2))/2;

AA(3:2:m-1,2:2:n)=(A2(3:2:m-1,2:2:n)+A2(3:2:m-1,1:2:n-1)+...
                                A2(2:2:m-2,2:2:n)+A2(2:2:m-2,1:2:n-1))/2;
AA(3:2:m-1,3:2:n-1)=(A2(3:2:m-1,3:2:n-1)+A2(2:2:m-2,3:2:n-1)+...
                                A2(3:2:m-1,2:2:n-2)+A2(2:2:m-2,2:2:n-2))/4;
AA(3:2:m-1,3:2:n-1)=AA(3:2:m-1,3:2:n-1)+(A1(3:2:m-1,3:2:n-1)+...
          A1(1:2:m-2,3:2:n-1)+A1(3:2:m-1,2:2:n-2)+A1(2:2:m-2,2:2:n-2))/4;
HH(2:m,2:n)=M(2:m,2:n)./AA(2:m,2:n);


Xc=sum(sum(M.*X))/MM;
Yc=sum(sum(M.*Y))/MM;
uc=sum(sum(M.*u))/MM;
vc=sum(sum(M.*v))/MM;
% This will do for now
xc=[xc,Xc];
yc=[yc,Yc];

if max(size(R))<100;
    R=[R,Xc^2+Yc^2];
    Time=[Time,t];
else
    R=[R(2:100),Xc^2+Yc^2];
    Time=[Time(2:100),t];
end

XX=X;
YY=Y;

ff=('velocity fields u-uc and v-vc -- both approximately planar fields');

ff1=['surface plots of water elevation h+(x^2+y^2)/2, the parabolic ...
                                boundary, and time  ' num2str(t)];
clf

subplot(2,2,1 )
surf(XX,YY,HH)
shading interp
hold on
contour3(XX,YY,HH,20,'k')
hold on
axis([-2 2 -2 2 0 2])
axis square
title('surface plot of water column above z=(x^2+y^2)/2')

subplot(322)
surf(XX-Xc,YY-Yc,u-uc)
shading interp
hold on
surf(XX-Xc,YY-Yc,v-vc)
shading interp
contour3(XX-Xc,YY-Yc,u-uc,20,'k')
title(ff)
axis([-1.5 1.5 -1.5 1.5 -1.5 1.5])% -.5 .5])
axis square



subplot(3,2,4)
plot(xc,yc,'r',Xc,Yc,'+k',0,0,'+k',[-2 2],[0 0],'k',[0 0],[-2 2],'k')
```

```
    axis([-1 1 -1 1])
    title('trajectory of the center of mass-initial position (1/sqrt(2) , 0) ')

    axis square

    subplot(223)
    ZZ1=((XX.^2+YY.^2)/2) ;
    ZZ2=HH+(XX.^2+YY.^2)/2;
    surf(XX,YY,ZZ1)
    shading interp
    hold on
    surf(XX,YY,ZZ2)
    shading interp
    hold on
    contour3(XX,YY,ZZ2,20,'k')
    hold on
    contour(XX,YY,ZZ2,20,'k')
    axis([-2 2 -2 2 0 2.5])
    axis square
    title(ff1)

    subplot(3,2,6)
    plot(Time,R)
    axis([min(Time),max(Time), min(R),max(R)])
    title('xc^2+yc^2 vs time')
    drawnow

    %pause(0.1)

end
```